

#### 9.5.9.1 Deploy a TiDB cluster of three data centers

For the three-data-center deployment method, refer to [Multiple Data Centers in One City Deployment](#).

Note that if both the TiKV and TiDB nodes have the configuration item `labels` configured, the TiKV and TiDB nodes in the same data center must have the same value for the `zone` label. For example, if a TiKV node and a TiDB node are both in the data center `dc-1`, then the two nodes need to be configured with the following label:

```
[labels]
zone=dc-1
```

#### 9.5.9.2 Perform local read using Stale Read

**Stale Read** is a mechanism that TiDB provides for the users to read historical data. Using this mechanism, you can read the corresponding historical data of a specific point in time or within a specified time range, and thus save the latency brought by data replication between storage nodes. When using Stale Read in some scenarios of geo-distributed deployment, TiDB accesses the replica in the current data center to read the corresponding data at the expense of some real-time performance, which avoids network latency brought by cross-center connection and reduces the access latency for the entire query process.

When TiDB receives a Stale Read query, if the `zone` label of that TiDB node is configured, then TiDB sends the request to the TiKV node with the same `zone` label where the corresponding data replica resides.

For how to perform Stale Read, see [Perform Stale Read using the AS OF TIMESTAMP clause](#).

## 9.6 Placement Rules

### Warning:

In the scenario of using TiFlash, the Placement Rules feature has been extensively tested and can be used in the production environment. Except for the scenario where TiFlash is used, using Placement Rules alone has not been extensively tested, so it is not recommended to enable this feature separately in the production environment.

### Note:

TiDB v5.3.0 introduces an experimental support for [Placement Rules in SQL](#). This offers a more convenient way to configure the placement of tables and partitions. Placement Rules in SQL might replace placement configuration with PD in future releases.

Placement Rules is an experimental feature of the Placement Driver (PD) introduced in v4.0. It is a replica rule system that guides PD to generate corresponding schedules for different types of data. By combining different scheduling rules, you can finely control the attributes of any continuous data range, such as the number of replicas, the storage location, the host type, whether to participate in Raft election, and whether to act as the Raft leader.

The Placement Rules feature is enabled by default in v5.0 and later versions of TiDB. To disable it, refer to [Disable Placement Rules](#).

### 9.6.1 Rule system

The configuration of the whole rule system consists of multiple rules. Each rule can specify attributes such as the number of replicas, the Raft role, the placement location, and the key range in which this rule takes effect. When PD is performing schedule, it first finds the rule corresponding to the Region in the rule system according to the key range of the Region, and then generates the corresponding schedule to make the distribution of the Region replica comply with the rule.

The key ranges of multiple rules can have overlapping parts, which means that a Region can match multiple rules. In this case, PD decides whether the rules overwrite each other or take effect at the same time according to the attributes of rules. If multiple rules take effect at the same time, PD will generate schedules in sequence according to the stacking order of the rules for rule matching.

In addition, to meet the requirement that rules from different sources are isolated from each other, these rules can be organized in a more flexible way. Therefore, the concept of “Group” is introduced. Generally, users can place rules in different groups according to different sources.

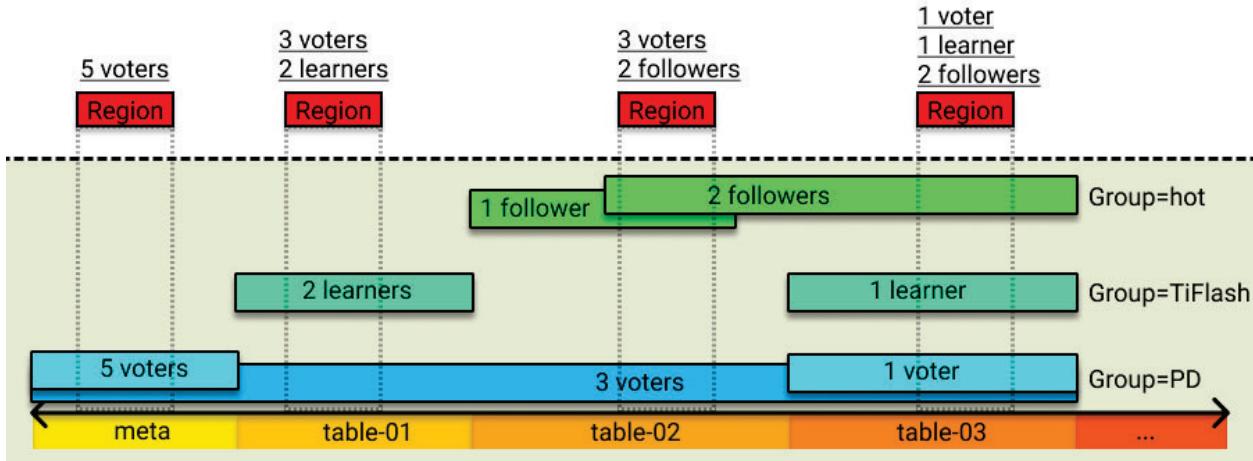


Figure 118: Placement rules overview

#### 9.6.1.1 Rule fields

The following table shows the meaning of each field in a rule:

Field name	Type and restriction	Description
GroupID	string	The group ID that marks the source of the rule.
ID	string	The unique ID of a rule in a group.
Index	int	The stacking sequence of rules in a group.
Override	true/false	Whether to overwrite rules with smaller index (in a group).
StartKey	string, in hexadecimal form	Applies to the starting key of a range.
EndKey	string, in hexadecimal form	Applies to the ending key of a range.
Role	string	Replica roles, including leader/follower/learner.
Count	int, positive integer	The number of replicas.
LabelConstraint	[] Constraint	Filters nodes based on the label.
LocationLabels	[] string	Used for physical isolation.
IsolationLevel	string	Used to set the minimum physical isolation level

`LabelConstraint` is similar to the function in Kubernetes that filters labels based on these four primitives: `in`, `notIn`, `exists`, and `notExists`. The meanings of these four primitives are as follows:

- `in`: the label value of the given key is included in the given list.
- `notIn`: the label value of the given key is not included in the given list.
- `exists`: includes the given label key.
- `notExists`: does not include the given label key.

The meaning and function of `LocationLabels` are the same with those earlier than v4.0. For example, if you have deployed `[zone, rack, host]` that defines a three-layer topology:

the cluster has multiple zones (Availability Zones), each zone has multiple racks, and each rack has multiple hosts. When performing schedule, PD first tries to place the Region's peers in different zones. If this try fails (such as there are three replicas but only two zones in total), PD guarantees to place these replicas in different racks. If the number of racks is not enough to guarantee isolation, then PD tries the host-level isolation.

The meaning and function of `IsolationLevel` is elaborated in [Cluster topology configuration](#). For example, if you have deployed `[zone,rack,host]` that defines a three-layer topology with `LocationLabels` and set `IsolationLevel` to `zone`, then PD ensures that all peers of each Region are placed in different zones during the scheduling. If the minimum isolation level restriction on `IsolationLevel` cannot be met (for example, 3 replicas are configured but there are only 2 data zones in total), PD will not try to make up to meet this restriction. The default value of `IsolationLevel` is an empty string, which means that it is disabled.

### 9.6.1.2 Fields of the rule group

The following table shows the description of each field in a rule group:

Field name	Type and restriction	Description
<code>ID</code>	<code>string</code>	The group ID that marks the source of the rule.
<code>Index</code>	<code>int</code>	The stacking sequence of different groups.
<code>Override</code>	<code>true/false</code>	Whether to override groups with smaller indexes.

## 9.6.2 Configure rules

The operations in this section are based on `pd-ctl`, and the commands involved in the operations also support calls via HTTP API.

### 9.6.2.1 Enable Placement Rules

The Placement Rules feature is enabled by default in v5.0 and later versions of TiDB. To disable it, refer to [Disable Placement Rules](#). To enable this feature after it has been disabled, you can modify the PD configuration file as follows before initializing the cluster:

```
[replication]
enable-placement-rules = true
```

In this way, PD enables this feature after the cluster is successfully bootstrapped and generates corresponding rules according to the `max-replicas` and `location-labels` configurations:

```
{
  "group_id": "pd",
  "id": "default",
  "start_key": "",
```

```

"end_key": "",
"role": "voter",
"count": 3,
"location_labels": ["zone", "rack", "host"],
"isolation_level": ""
}

```

For a bootstrapped cluster, you can also enable Placement Rules online through pd-ctl:

```
pd-ctl config placement-rules enable
```

PD also generates default rules based on the `max-replicas` and `location-labels` configurations.

#### Note:

After enabling Placement Rules, the previously configured `max-replicas` and `location-labels` no longer take effect. To adjust the replica policy, use the interface related to Placement Rules.

#### 9.6.2.2 Disable Placement Rules

You can use pd-ctl to disable the Placement Rules feature and switch to the previous scheduling strategy.

```
pd-ctl config placement-rules disable
```

#### Note:

After disabling Placement Rules, PD uses the original `max-replicas` and `location-labels` configurations. The modification of rules (when Placement Rules is enabled) will not update these two configurations in real time. In addition, all the rules that have been configured remain in PD and will be used the next time you enable Placement Rules.

#### 9.6.2.3 Set rules using pd-ctl

**Note:**

The change of rules affects the PD scheduling in real time. Improper rule setting might result in fewer replicas and affect the high availability of the system.

pd-ctl supports using the following methods to view rules in the system, and the output is a JSON-format rule or a rule list.

- To view the list of all rules:

```
pd-ctl config placement-rules show
```

- To view the list of all rules in a PD Group:

```
pd-ctl config placement-rules show --group=pd
```

- To view the rule of a specific ID in a Group:

```
pd-ctl config placement-rules show --group=pd --id=default
```

- To view the rule list that matches a Region:

```
pd-ctl config placement-rules show --region=2
```

In the above example, 2 is the Region ID.

Adding rules and editing rules are similar. You need to write the corresponding rules into a file and then use the `save` command to save the rules to PD:

```
cat > rules.json <<EOF
[
  {
    "group_id": "pd",
    "id": "rule1",
    "role": "voter",
    "count": 3,
    "location_labels": ["zone", "rack", "host"]
  },
  {
    "group_id": "pd",
    "id": "rule2",
    "role": "voter",
    "count": 2,
  }
]
```

```

        "location_labels": ["zone", "rack", "host"]
    }
]
EOF
pd-ctl config placement save --in=rules.json

```

The above operation writes rule1 and rule2 to PD. If a rule with the same GroupID + ID already exists in the system, this rule is overwritten.

To delete a rule, you only need to set the count of the rule to 0, and the rule with the same GroupID + ID will be deleted. The following command deletes the pd / rule2 rule:

```

cat > rules.json <<EOF
[
  {
    "group_id": "pd",
    "id": "rule2"
  }
]
EOF
pd-ctl config placement save --in=rules.json

```

#### 9.6.2.4 Use pd-ctl to configure rule groups

- To view the list of all rule groups:

```
pd-ctl config placement-rules rule-group show
```

- To view the rule group of a specific ID:

```
pd-ctl config placement-rules rule-group show pd
```

- To set the index and override attributes of the rule group:

```
pd-ctl config placement-rules rule-group set pd 100 true
```

- To delete the configuration of a rule group (use the default group configuration if there is any rule in the group):

```
pd-ctl config placement-rules rule-group delete pd
```

### 9.6.2.5 Use pd-ctl to batch update groups and rules in groups

To view and modify the rule groups and all rules in the groups at the same time, execute the `rule-bundle` subcommand.

In this subcommand, `get {group_id}` is used to query a group, and the output result shows the rule group and rules of the group in a nested form:

```
pd-ctl config placement-rules rule-bundle get pd
```

The output of the above command:

```
{
  "group_id": "pd",
  "group_index": 0,
  "group_override": false,
  "rules": [
    {
      "group_id": "pd",
      "id": "default",
      "start_key": "",
      "end_key": "",
      "role": "voter",
      "count": 3
    }
  ]
}
```

To write the output to a file, add the `-out` argument to the `rule-bundle get` subcommand, which is convenient for subsequent modification and saving.

```
pd-ctl config placement-rules rule-bundle get pd -out="group.json"
```

After the modification is finished, you can use the `rule-bundle set` subcommand to save the configuration in the file to the PD server. Unlike the `save` command described in [Set rules using pd-ctl](#), this command replaces all the rules of this group on the server side.

```
pd-ctl config placement-rules rule-bundle set pd -in="group.json"
```

### 9.6.2.6 Use pd-ctl to view and modify all configurations

You can also view and modify all configuration using pd-ctl. To do that, save all configuration to a file, edit the configuration file, and then save the file to the PD server to overwrite the previous configuration. This operation also uses the `rule-bundle` subcommand.

For example, to save all configuration to the `rules.json` file, execute the following command:

```
pd-ctl config placement-rules rule-bundle load --out="rules.json"
```

After editing the file, execute the following command to save the configuration to the PD server:

```
pd-ctl config placement-rules rule-bundle save --in="rules.json"
```

#### 9.6.2.7 Use tidb-ctl to query the table-related key range

If you need special configuration for metadata or a specific table, you can execute the `keyrange` command in `tidb-ctl` to query related keys. Remember to add `--encode` at the end of the command.

```
tidb-ctl keyrange --database test --table ttt --encode
```

```
global ranges:
meta: (6d00000000000000f8, 6e00000000000000f8)
table: (7400000000000000f8, 7500000000000000f8)
table ttt ranges: (NOTE: key range might be changed after DDL)
table: (74800000000000ff2d000000000000f8, 7480000000000000
    ↳ ff2e00000000000000f8)
table indexes: (74800000000000ff2d5f69000000000fa, 7480000000000000
    ↳ ff2d5f720000000000fa)
index c2: (74800000000000ff2d5f69800000000ff000010000000000fa,
    ↳ 74800000000000ff2d5f69800000000ff000020000000000fa)
index c3: (74800000000000ff2d5f69800000000ff000020000000000fa,
    ↳ 74800000000000ff2d5f69800000000ff000030000000000fa)
index c4: (74800000000000ff2d5f69800000000ff000030000000000fa,
    ↳ 74800000000000ff2d5f69800000000ff000040000000000fa)
table rows: (74800000000000ff2d5f72000000000fa, 7480000000000000
    ↳ ff2e00000000000000f8)
```

#### Note:

DDL and other operations can cause table ID changes, so you need to update the corresponding rules at the same time.

#### 9.6.3 Typical usage scenarios

This section introduces the typical usage scenarios of Placement Rules.

##### 9.6.3.1 Scenario 1: Use three replicas for normal tables and five replicas for the metadata to improve cluster disaster tolerance

You only need to add a rule that limits the key range to the range of metadata, and set the value of `count` to 5. Here is an example of this rule:

```
{
    "group_id": "pd",
    "id": "meta",
    "index": 1,
    "override": true,
    "start_key": "6d00000000000000f8",
    "end_key": "6e00000000000000f8",
    "role": "voter",
    "count": "5",
    "location_labels": ["zone", "rack", "host"]
}
```

#### 9.6.3.2 Scenario 2: Place five replicas in three data centers in the proportion of 2:2:1, and the Leader should not be in the third data center

Create three rules. Set the number of replicas to 2, 2, and 1 respectively. Limit the replicas to the corresponding data centers through `label_constraints` in each rule. In addition, change `role` to `follower` for the data center that does not need a Leader.

```
[
    {
        "group_id": "pd",
        "id": "zone1",
        "start_key": "",
        "end_key": "",
        "role": "voter",
        "count": 2,
        "label_constraints": [
            {"key": "zone", "op": "in", "values": ["zone1"]}
        ],
        "location_labels": ["rack", "host"]
    },
    {
        "group_id": "pd",
        "id": "zone2",
        "start_key": "",
        "end_key": "",
        "role": "voter",
        "count": 2,
        "label_constraints": [
            {"key": "zone", "op": "in", "values": ["zone2"]}
        ],
        "location_labels": ["rack", "host"]
    },
    {
        "group_id": "pd",
        "id": "zone3",
        "start_key": "",
        "end_key": "",
        "role": "follower",
        "count": 1,
        "label_constraints": [
            {"key": "zone", "op": "in", "values": ["zone3"]}
        ],
        "location_labels": ["rack", "host"]
    }
]
```

```

},
{
    "group_id": "pd",
    "id": "zone3",
    "start_key": "",
    "end_key": "",
    "role": "follower",
    "count": 1,
    "label_constraints": [
        {"key": "zone", "op": "in", "values": ["zone3"]}
    ],
    "location_labels": ["rack", "host"]
}
]

```

#### 9.6.3.3 Scenario 3: Add two TiFlash replicas for a table

Add a separate rule for the row key of the table and limit count to 2. Use `label_constraints` to ensure that the replicas are generated on the node of `engine ↪ = tiflash`. Note that a separate `group_id` is used here to ensure that this rule does not overlap or conflict with rules from other sources in the system.

```
{
    "group_id": "tiflash",
    "id": "learner Replica-table-ttt",
    "start_key": "748000000000000ff2d5f72000000000fa",
    "end_key": "748000000000000ff2e0000000000000f8",
    "role": "learner",
    "count": 2,
    "label_constraints": [
        {"key": "engine", "op": "in", "values": ["tiflash"]}
    ],
    "location_labels": ["host"]
}
```

#### 9.6.3.4 Scenario 4: Add two follower replicas for a table in the Beijing node with high-performance disks

The following example shows a more complicated `label_constraints` configuration. In this rule, the replicas must be placed in the `bj1` or `bj2` machine room, and the disk type must not be `hdd`.

```
{
    "group_id": "follower-read",
    "id": "follower-read-table-ttt",

```

```

"start_key": "748000000000000ff2d000000000000f8",
"end_key": "748000000000000ff2e000000000000f8",
"role": "follower",
"count": 2,
"label_constraints": [
    {"key": "zone", "op": "in", "values": ["bj1", "bj2"]},
    {"key": "disk", "op": "notIn", "values": ["hdd"]}
],
"location_labels": ["host"]
}

```

#### 9.6.3.5 Scenario 5: Migrate a table to the TiFlash cluster

Different from scenario 3, this scenario is not to add new replica(s) on the basis of the existing configuration, but to forcibly override the other configuration of a data range. So you need to specify an `index` value large enough and set `override` to `true` in the rule group configuration to override the existing rule.

The rule:

```
{
  "group_id": "tiflash-override",
  "id": "learner Replica-table-ttt",
  "start_key": "748000000000000ff2d5f7200000000fa",
  "end_key": "748000000000000ff2e000000000000f8",
  "role": "voter",
  "count": 3,
  "label_constraints": [
    {"key": "engine", "op": "in", "values": ["tiflash"]}
  ],
  "location_labels": ["host"]
}
```

The rule group:

```
{
  "id": "tiflash-override",
  "index": 1024,
  "override": true,
}
```

## 9.7 Load Base Split

Load Base Split is a new feature introduced in TiDB 4.0. It aims to solve the hotspot issue caused by unbalanced access between Regions, such as full table scans for small tables.

### 9.7.1 Scenarios

In TiDB, it is easy to generate hotspots when the load is concentrated on certain nodes. PD tries to schedule the hot Regions so that they are distributed as evenly as possible across all nodes for better performance.

However, the minimum unit for PD scheduling is Region. If the number of hotspots in a cluster is smaller than the number of nodes, or if a few hotspots have far more load than other Regions, PD can only move the hotspot from one node to another, but not make the entire cluster share the load.

This scenario is especially common with workloads that are mostly read requests, such as full table scans and index lookups for small tables, or frequent access to some fields.

Previously, the solution to this problem was to manually execute a command to split one or more hotspot Regions, but this approach has two problems:

- Evenly splitting a Region is not always the best choice, because requests might be concentrated on a few keys. In such cases, hotspots might still be on one of the Regions after evenly splitting, and it might take multiple even splits to realize the goal.
- Human intervention is not timely or simple.

### 9.7.2 Implementation principles

Load Base Split automatically splits the Region based on statistics. It identifies the Regions whose read load consistently exceeds the threshold for 10 seconds, and splits these Regions at a proper position. When choosing the split position, Load Base Split tries to balance the access load of both Regions after the split and avoid access across Regions.

The Region split by Load Base Split will not be merged quickly. On the one hand, PD's `MergeChecker` skips the hot Regions; on the other hand, PD also determines whether to merge two Regions according to QPS in the heartbeat information, to avoid the merging of two Regions with high QPS.

### 9.7.3 Usage

The Load Base Split feature is currently controlled by the `split.qps-threshold` parameter (QPS threshold) and `split.byte-threshold` parameter (traffic threshold). If the sum of all types of read requests per second for a Region exceeds the QPS threshold or traffic threshold for 10 consecutive seconds, PD splits the Region.

Load Base Split is enabled by default, but the parameter is set to a rather high value. `split.qps-threshold` defaults to 3000 and `split.byte-threshold` defaults to 30MB/s. If you want to disable this feature, set the two thresholds high enough at the same time.

To modify the parameter, take either of the following two methods:

- Use a SQL statement:

```
set config tikv split.qps-threshold=3000
```

- Use TiKV:

```
curl -X POST "http://ip:status_port/config" -H "accept: application/json" -d '{"split.qps-threshold":"3000"}'
```

Accordingly, you can view the configuration by either of the following two methods:

- Use a SQL statement:

```
show config where type='tikv' and name like '%split.qps-threshold%'
```

- Use TiKV:

```
curl "http://ip:status_port/config"
```

#### Note:

Starting from v4.0.0-rc.2, you can modify and view the configuration using SQL statements.

## 9.8 Store Limit

Store Limit is a feature of PD, introduced in TiDB 3.0. It is designed to control the scheduling speed in a finer manner for better performance in different scenarios.

### 9.8.1 Implementation principles

PD performs scheduling at the unit of operator. An operator might contain several scheduling operations. For example:

```
"replace-down-replica {mv peer: store [2] to [3]} (kind:region,replica,
    ↪ region:10(4,5), createAt:2020-05-18 06:40:25.775636418 +0000 UTC m
    ↪ =+2168762.679540369, startAt:2020-05-18 06:40:25.775684648 +0000 UTC
    ↪ m=+2168762.679588599, currentStep:0, steps:[add learner peer 20 on
    ↪ store 3, promote learner peer 20 on store 3 to voter, remove peer on
    ↪ store 2])"
```

In this above example, the `replace-down-replica` operator contains the following specific operations:

1. Add a learner peer with the ID 20 to `store 3`.
2. Promote the learner peer with the ID 20 on `store 3` to a voter.
3. Delete the peer on `store 2`.

Store Limit achieves the store-level speed limit by maintaining a mapping from store IDs to token buckets in memory. The different operations here correspond to different token buckets. Currently, Store Limit only supports limiting the speed of two operations: adding learners/peers and deleting peers. That is, each store has two types of token buckets.

Every time an operator is generated, it checks whether enough tokens exist in the token buckets for its operations. If yes, the operator is added to the scheduling queue, and the corresponding token is taken from the token bucket. Otherwise, the operator is abandoned. Because the token bucket replenishes tokens at a fixed rate, the speed limit is thus achieved.

Store Limit is different from other limit-related parameters in PD (such as `region-schedule-limit` and `leader-schedule-limit`) in that it mainly limits the consuming speed of operators, while other parameters limits the generating speed of operators. Before introducing the Store Limit feature, the speed limit of scheduling is mostly at the global scope. Therefore, even if the global speed is limited, it is still possible that the scheduling operations are concentrated on some stores, affecting the performance of the cluster. By limiting the speed at a finer level, Store Limit can better control the scheduling behavior.

## 9.8.2 Usage

The parameters of Store Limit can be configured using `pd-ctl`.

### 9.8.2.1 View setting of the current store

To view the limit setting of the current store, run the following commands:

```
store limit          // Shows the speed limit of adding and
    ↪ deleting peers in all stores.
store limit add-peer // Shows the speed limit of adding peers in
    ↪ all stores.
store limit remove-peer // Shows the speed limit of deleting peers
    ↪ in all stores.
```

### 9.8.2.2 Set limit for all stores

To set the speed limit for all stores, run the following commands:

```

store limit all 5          // All stores can at most add and delete 5
  ↢ peers per minute.
store limit all 5 add-peer // All stores can at most add 5 peers per
  ↢ minute.
store limit all 5 remove-peer // All stores can at most delete 5 peers per
  ↢ minute.

```

### 9.8.2.3 Set limit for a single store

To set the speed limit for a single store, run the following commands:

```

store limit 1 5          // store 1 can at most add and delete 5
  ↢ peers per minute.
store limit 1 5 add-peer // store 1 can at most add 5 peers per
  ↢ minute.
store limit 1 5 remove-peer // store 1 can at most delete 5 peers per
  ↢ minute.

```

## 10 TiDB Tools

### 10.1 TiDB Tools Overview

TiDB provides a rich set of tools to help you with deployment operations, data management (such as import and export, data migration, backup & recovery), and complex OLAP queries. You can select the applicable tools according to your needs.

#### 10.1.1 Deployment and operation Tools

To meet your deployment and operation needs in different system environments, TiDB provides two deployment and Operation tools, TiUP and TiDB Operator.

##### 10.1.1.1 Deploy and operate TiDB on physical or virtual machines

[TiUP](#) is a TiDB package manager on physical or virtual machines. TiUP can manage multiple TiDB components such as TiDB, PD, TiKV. To start any component in the TiDB ecosystem, you just need to execute a single TiUP command.

TiUP provides [TiUP cluster](#), a cluster management component written in Golang. By using TiUP cluster, you can easily perform daily database operations, including deploying, starting, stopping, destroying, scaling, and upgrading a TiDB cluster, and manage TiDB cluster parameters.

The following are the basics of TiUP:

- Terminology and Concepts
- Deploy a TiDB Cluster Using TiUP
- Manage TiUP Components with TiUP Commands
- Applicable TiDB versions: v4.0 and above

#### 10.1.1.2 Deploy and operate TiDB in Kubernetes

TiDB Operator is an automatic operation system for TiDB clusters in Kubernetes. It provides full life-cycle management for TiDB including deployment, upgrades, scaling, backup, fail-over, and configuration changes. With TiDB Operator, TiDB can run seamlessly in the Kubernetes clusters deployed on a public or private cloud.

The following are the basics of TiDB Operator:

- TiDB Operator Architecture
- Get Started with TiDB Operator in Kubernetes
- Applicable TiDB versions: v2.1 and above

#### 10.1.2 Data management tools

TiDB provides multiple data management tools, such as import and export, backup and restore, data replication, data migration, incremental synchronization, and data validation.

##### 10.1.2.1 Full data export

Dumpling is a tool for the logical full data export from MySQL or TiDB.

The following are the basics of Dumpling:

- Input: MySQL/TiDB cluster
- Output: SQL/CSV file
- Supported TiDB versions: all versions
- Kubernetes support: No

##### Note:

PingCAP previously maintained a fork of the [mydumper project](#) with enhancements specific to TiDB. This fork has since been replaced by [Dumpling](#), which has been rewritten in Go, and supports more optimizations that are specific to TiDB. It is strongly recommended that you use Dumpling instead of mydumper.

### 10.1.2.2 Full data import

TiDB Lightning (Lightning) is a tool used for the full import of large amounts of data into a TiDB cluster. Currently, TiDB Lightning supports reading SQL dump exported via Dumpling or CSV data source.

TiDB Lightning supports three modes:

- **local**: TiDB Lightning parses data into ordered key-value pairs and directly imports them into TiKV. This mode is usually for importing a large amount of data (at the TB level) to a new cluster. During the import, the cluster cannot provide services.
- **importer**: This mode is similar to the **local** mode. To use this mode, you need to deploy an additional component `tikv-importer` to help import key-value pairs. If the target cluster is in v4.0 or later versions, it is recommended to use the **local** mode.
- **tidb**: This mode uses TiDB/MySQL as the backend, which is slower than the **local** mode and **importer** mode but can be performed online. It also supports importing data to MySQL.

The following are the basics of TiDB Lightning:

- Input data source:
  - The output file of Dumpling
  - Other compatible CSV file
- Supported TiDB versions: v2.1 or later
- Kubernetes support: Yes. See [Quickly restore data into a TiDB cluster in Kubernetes using TiDB Lightning](#) for details.

#### Note:

The Loader tool is no longer maintained. For scenarios related to Loader, it is recommended that you use the `tidb` mode of TiDB Lightning instead. For details, see [TiDB Lightning TiDB backends](#).

### 10.1.2.3 Backup and restore

Backup & Restore (BR) is a command-line tool for distributed backup and restore of the TiDB cluster data. BR can effectively back up and restore TiDB clusters of huge data volume.

The following are the basics of BR:

- **Input and output data source**: SST + `backupmeta` file
- Supported TiDB versions: v3.1 and v4.0
- Kubernetes support: Yes. See [Back up Data to S3-Compatible Storage Using BR](#) and [Restore Data from S3-Compatible Storage Using BR](#) for details.

#### 10.1.2.4 Incremental data replication

[TiDB Binlog](#) is a tool that collects binlog for TiDB clusters and provides near real-time sync and backup. It can be used for incremental data replication between TiDB clusters, such as making a TiDB cluster the secondary cluster of the primary TiDB cluster.

The following are the basics of TiDB Binlog:

- Input/Output:
  - Input: TiDB cluster
  - Output: TiDB cluster, MySQL, Kafka or incremental backup files
- Supported TiDB versions: v2.1 or later
- Kubernetes support: Yes. See [TiDB Binlog Cluster Operations](#) and [TiDB Binlog Drainer Configurations in Kubernetes](#) for details.

#### 10.1.2.5 Data migration

[TiDB Data Migration](#) (DM) is an integrated data replication task management platform that supports the full data migration and the incremental data replication from MySQL/-MariaDB to TiDB.

The following are the basics of DM:

- Input: MySQL/MariaDB
- Output: TiDB cluster
- Supported TiDB versions: all versions
- Kubernetes support: No, under development

If the data volume is below the TB level, it is recommended to migrate data from MySQL/MariaDB to TiDB directly using DM. The migration process includes the full data import and export and the incremental data replication.

If the data volume is at the TB level, take the following steps:

1. Use [Dumpling](#) to export the full data from MySQL/MariaDB.
2. Use [TiDB Lightning](#) to import the data exported in Step 1 to the TiDB cluster.
3. Use DM to replicate the incremental data from MySQL/MariaDB to TiDB.

#### Note:

The Syncer tool is no longer maintained. For scenarios related to Syncer, it is recommended that you use DM's incremental task mode instead.

### 10.1.3 OLAP Query tool

TiDB provides the OLAP query tool TiSpark, which allows you to query TiDB tables as if you were using native Spark.

#### 10.1.3.1 Query TiKV data source using Spark

**TiSpark** is a thin layer built for running Apache Spark on top of TiKV to answer the complex OLAP queries. It takes advantages of both the Spark platform and the distributed TiKV cluster and seamlessly glues to TiDB, and provides a one-stop Hybrid Transactional and Analytical Processing (HTAP) solution.

## 10.2 TiDB Tools Use Cases

This document introduces the common use cases of TiDB tools and how to choose the right tool for your scenario.

### 10.2.1 Deploy and operate TiDB on physical or virtual machines

If you need to deploy and operate TiDB on physical or virtual machines, you can install **TiUP**, and then use TiUP to manage TiDB components such as TiDB, PD, and TiKV.

### 10.2.2 Deploy and operate TiDB in Kubernetes

If you need to deploy and operate TiDB in Kubernetes, you can deploy a Kubernetes cluster, and then deploy **TiDB Operator**. After that, you can use TiDB Operator to deploy and operate a TiDB cluster.

### 10.2.3 Import data from CSV to TiDB

If you need to import the compatible CSV files exported by other tools to TiDB, use **TiDB Lightning**.

### 10.2.4 Import full data from MySQL/Aurora

If you need to import full data from MySQL/Aurora, use **Dumpling** first to export data as SQL dump files, and then use **TiDB Lightning** to import data into the TiDB cluster.

### 10.2.5 Migrate data from MySQL/Aurora

If you need to migrate both full data and incremental data from MySQL/Aurora, use **TiDB Data Migration (DM)** to perform the **full and incremental data migration**.

If the full data volume is large (at the TB level), you can first use [Dumpling](#) and [TiDB Lightning](#) to perform the full data migration, and then use DM to perform the incremental data migration.

#### 10.2.6 Back up and restore TiDB cluster

If you need to back up a TiDB cluster or restore backed up data to the cluster, use [BR](#) (Backup & Restore).

In addition, BR can also be used to perform [incremental backup](#) and [incremental restore](#) of TiDB cluster data.

#### 10.2.7 Migrate data to TiDB

If you need to migrate data from a TiDB cluster to another TiDB cluster, use [Dumpling](#) to export full data from TiDB as SQL dump files, and then use [TiDB Lightning](#) to import data to another TiDB cluster.

If you also need to migrate incremental data, use [TiDB Binlog](#).

#### 10.2.8 TiDB incremental data subscription

If you need to subscribe to TiDB's incremental changes, use [TiDB Binlog](#).

### 10.3 Download TiDB Tools

This document collects the available downloads for most officially maintained versions of TiDB tools.

#### 10.3.1 TiUP

You can install TiUP with a single command in both Darwin and Linux operating systems. For more information, see [Install TiUP](#).

#### 10.3.2 TiDB Operator

TiDB Operator runs in Kubernetes. After deploying the Kubernetes cluster, you can choose to deploy TiDB Operator either online or offline. For more information, see [Deploying TiDB Operator in Kubernetes](#).

#### 10.3.3 TiDB Binlog

If you want to download the latest version of [TiDB Binlog](#), directly download the TiDB package, because TiDB Binlog is included in the TiDB package.

Package name	OS	Architecture	SHA256
check-			
<a href="https://download.pingcap.org/">https://download.pingcap.org/</a>	Linux	amd64	<a href="https://pingcap.github.io/tidb-binlog/v5.3.0/tidb-v5.3.0-linux-amd64.tar.gz.sha256">https://pingcap.github.io/tidb-binlog/v5.3.0/tidb-v5.3.0-linux-amd64.tar.gz.sha256</a>
→ .			→ .
→ pingcap			→ pingcap
→ .			→ .
→ org			→ org
→ /			→ /
→ tidb			→ tidb
→ -{			→ -{
→ version			→ version
→ }-			→ }-
→ linux			→ linux
→ -			→ -
→ amd64			→ amd64
→ .			→ .
→ tar			→ sha256
→ .			→
→ gz			
→			
(TiDB Bin-log)			

#### Note:

{version} in the above download link indicates the version number of TiDB. For example, the download link for v5.3.0 is <https://download.pingcap.org/org/tidb-v5.3.0-linux-amd64.tar.gz>.

#### 10.3.4 TiDB Lightning

Download [TiDB Lightning](#) by using the download link in the following table:

Package name	OS	Architecture	SHA256
check-			
<a href="https://download.pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz">https://download.pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz</a>	Linux	amd64	https://pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz
→ https://			→ https://pingcap.org/
→ download			→ download
→ .			→ .
→ pingcap			→ pingcap
→ .			→ .
→ org			→ org
→ /			→ /
→ tidb			→ tidb
→ -			→ -
→ toolkit			→ toolkit
→ -{			→ -{
→ version			→ version
→ }-			→ }-
→ linux			→ linux
→ -			→ -
→ amd64			→ amd64
→ .			→ .
→ tar			→ sha256
→ .			→
→ gz			
→			

#### Note:

{version} in the above download link indicates the version number of TiDB Lightning. For example, the download link for v5.3.0 is <https://download.pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz>.

#### 10.3.5 BR (backup and restore)

Download **BR** by using the download link in the following table:

Package name	OS	Architecture	SHA256
checksum			
http	Linux	amd64	http
→ ://			→ ://
→ download			→ download
→ .			→ .
→ pingcap			→ pingcap
→ .			→ .
→ org			→ org
→ /			→ /
→ tidb			→ tidb
→ -			→ -
→ toolkit			→ toolkit
→ -{			→ -{
→ version			→ version
→ }-			→ }-
→ linux			→ linux
→ -			→ -
→ amd64			→ amd64
→ .			→ .
→ tar			→ sha256
→ .			→
→ gz			
→			

#### Note:

{version} in the above download link indicates the version number of BR. For example, the download link for v5.3.0 is <https://download.pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz>.

### 10.3.6 TiDB DM (Data Migration)

Download [DM](#) by using the download link in the following table:

Package name	OS	Architecture	SHA256
check-			
<a href="https://download.pingcap.org/">https://download.pingcap.org/</a>	Linux	amd64	<a href="https://pingcap.fasrc01.pingcap.com/tar/sha256sums.sha256">https://pingcap.fasrc01.pingcap.com/tar/sha256sums.sha256</a>
→ .			→ .
→ pingcap			→ pingcap
→ .			→ .
→ org			→ org
→ /			→ /
→ dm			→ dm
→ -{			→ -{
→ version			→ version
→ }-			→ }-
→ linux			→ linux
→ -			→ -
→ amd64			→ amd64
→ .			→ .
→ tar			→ sha256
→ .			→
→ gz			
→			

#### Note:

{version} in the above download link indicates the version number of DM. For example, the download link for v5.3.0 is <https://download.pingcap.org/org/dm-v5.3.0-linux-amd64.tar.gz>. You can check the published DM versions in the [DM Release](#) page.

#### 10.3.7 Dumpling

Download [Dumpling](#) from the links below:

Install	Operating	SHA256
pack-	sys-	check-
age	tem	Architecture
<a href="https://download.pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz">https://download.pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz</a>	Linux	amd64
→ ://		https://
→ download		download
→ .		.
→ pingcap		pingcap
→ .		.
→ org		org
→ /		/
→ tidb		tidb
→ -		-
→ toolkit		toolkit
→ -{		-{
→ version		version
→ }-		}-
→ linux		linux
→ -		-
→ amd64		amd64
→ .		.
→ tar		sha256
→ .		
→ gz		
→		

#### Note:

The {version} in the download link is the version number of Dumpling. For example, the link for downloading the v5.3.0 version of Dumpling is <https://download.pingcap.org/tidb-toolkit-v5.3.0-linux-amd64.tar.gz>. You can view the currently released versions in [TiDB Releases](#).

Dumpling supports arm64 linux. You can replace `amd64` in the download link with `arm64`, which means the `arm64` version of Dumpling.

#### 10.3.8 sync-diff-inspector

Download [sync-diff-inspector](#) from the links below:

Package name	OS	Architecture	SHA256
			checksum
tidb-enterprise-tools-nightly-linux-amd64.tar.gz	Linux	amd64	tidb-enterprise-tools-nightly-linux-amd64.sha256

### 10.3.9 TiCDC

To download [TiCDC](#), refer to [Deploy TiCDC](#).

## 10.4 TiUP

### 10.4.1 TiUP Documentation Map

#### 10.4.1.1 User guide

- [TiUP Overview](#): Gives an overall introduction to TiUP, for example, how to install and use TiUP, and the related terminologies.
- [TiUP Terminology and Concepts](#): Explains the terms that you might bump into when using TiUP, and help you understand the key concepts of TiUP
- [TiUP Component Management](#): Introduces all TiUP commands in detail, and how to use TiUP to download, update and delete components
- [TiUP FAQ](#): Introduces common issues when you use TiUP, including FAQs of the third-party components of TiUP
- [TiUP Troubleshooting Guide](#): Introduces the troubleshooting methods and solutions if you encounter issues when using TiUP
- [TiUP Reference Guide](#): Introduces detailed references, including commands, components, and mirrors.

#### 10.4.1.2 TiUP resources

- [TiUP Issues](#): Lists TiUP GitHub issues

### 10.4.2 TiUP Overview

Starting with TiDB 4.0, TiUP, as the package manager, makes it far easier to manage different cluster components in the TiDB ecosystem. Now you can run any component with only a single line of TiUP commands.

#### 10.4.2.1 Install TiUP

You can install TiUP with a single command in both Darwin and Linux operating systems:

```
curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/
→ install.sh | sh
```

This command installs TiUP in the `$HOME/.tiup` folder. The installed components and the data generated by their operation are also placed in this folder. This command also automatically adds `$HOME/.tiup/bin` to the PATH environment variable in the Shell `.profile` file, so you can use TiUP directly.

After installation, you can check the version of TiUP:

```
tiup --version
```

#### Note:

By default, TiUP shares usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see [Telemetry](#).

#### 10.4.2.2 TiUP ecosystem introduction

TiUP is not only a package manager in the TiDB ecosystem. Its ultimate mission is to enable everyone to use TiDB ecosystem tools **easier than ever** by building its own ecosystem. This requires introducing additional packages to enrich the TiUP ecosystem.

This series of TiUP documents introduce what these packages do and how you can use them.

In the TiUP ecosystem, you can get help information by adding `--help` to any command, such as the following command to get help information for TiUP itself:

```
tiup --help
```

```
TiUP is a command-line component management tool that can help to download
→ and install
TiDB platform components to the local system. You can run a specific version
→ of a component via
"tiup <component>[:version]". If no version number is specified, the latest
→ version installed
locally will be used. If the specified component does not have any version
→ installed locally,
the latest stable version will be downloaded from the repository.
```

Usage:

```
tiup [flags] <command> [args...]
tiup [flags] <component> [args...]
```

Available Commands:

install	Install a specific version of a component
list	List the available TiDB components or versions
uninstall	Uninstall components or versions of a component
update	Update tiup components to the latest version
status	List the status of instantiated components
clean	Clean the data of instantiated components
mirror	Manage a repository mirror for TiUP components
help	Help about any command or component

Components Manifest:

```
use "tiup list" to fetch the latest components manifest
```

Flags:

-B, --binary <component>[:version]	Print binary path of a specific version of a component <component>[:version] and the latest version installed will be selected if no version specified
--binpath string	Specify the binary path of component instance
-h, --help	help for tiup
--skip-version-check	Skip the strict version check, by default a version must be a valid SemVer string
-T, --tag string	Specify a tag for component instance
-v, --version	version for tiup

Component instances with the same "tag" will share a data directory (

$\hookrightarrow$  \$TIUP\_HOME/data/\$tag):

```
$ tiup --tag mycluster playground
```

Examples:

\$ tiup playground	# Quick start
\$ tiup playground nightly	# Start a playground with the latest nightly version
\$ tiup install <component>[:version]	# Install a component of specific version
\$ tiup update --all	# Update all installed components to the latest version
\$ tiup update --nightly	# Update all installed components to the nightly version

```

$ tiup update --self          # Update the "tiup" to the latest version
$ tiup list                   # Fetch the latest supported components
    ↪ list
$ tiup status                 # Display all running/terminated
    ↪ instances
$ tiup clean <name>          # Clean the data of running/terminated
    ↪ instance (Kill process if it's running)
$ tiup clean --all            # Clean the data of all running/
    ↪ terminated instances

```

Use "tiup [command] --help" for more information about a command.

The output is long but you can focus on only two parts:

- Available commands
  - install: used to install components
  - list: used to view the list of available components
  - uninstall: used to uninstall components
  - update: used to update the component version
  - status: used to view the running history of components
  - clean: used to clear the running log of components
  - mirror: used to clone a private mirror from the official mirror
  - help: used to print out help information
- Available components
  - playground: used to start a TiDB cluster locally
  - client: used to connect to a TiDB cluster in a local machine
  - cluster: used to deploy a TiDB cluster for production environments
  - bench: used to stress test the database

#### Note:

- The number of available components will continue to grow. To check the latest supported components, execute the `tiup list` command.
- The list of available versions of components will also continue to grow. To check the latest supported component versions, execute the `tiup ↪ list <component>` command.

TiUP commands are implemented in TiUP's internal code and used for package management operations, while TiUP components are independent component packages installed by TiUP commands.

For example, if you run the `tiup list` command, TiUP directly runs its own internal code; if you run the `tiup playground` command, TiUP first checks whether there is a local package named “playground”, and if not, TiUP downloads the package from the mirror, and then run it.

### 10.4.3 TiUP Terminology and Concepts

This document explains important terms and concepts of TiUP.

#### 10.4.3.1 TiUP components

The TiUP program contains only a few commands for downloading, updating, and uninstalling components. TiUP expands its functions with various components. A **component** is a program or script that can be run. When running a component through `tiup <component>`, TiUP adds a set of environment variables, creates the data directory for the program, and then runs the program.

By running the `tiup <component>` command, you can run a component supported by TiUP. The running logic is:

- If you specify a version of a component through `tiup <component>[:version]`:
  - If the component does not have any version installed locally, TiUP downloads the latest stable version from the mirror server.
  - If the component has one or more versions installed locally, but there is no version specified by you, TiUP downloads the specified version from the mirror server.
  - If the specified version of the component is installed locally, TiUP sets the environment variable to run the installed version.
- If you run a component through `tiup <component>` and specify no version:
  - If the component does not have any version installed locally, TiUP downloads the latest stable version from the mirror server.
  - If one or more versions have been installed locally, TiUP sets the environment variable to run the latest installed version.

#### 10.4.3.2 TiUP mirrors

All components of TiUP are downloaded from the TiUP mirrors. TiUP mirrors contain the TAR package of each component and the corresponding meta information (version, entry startup file, checksum). TiUP uses PingCAP’s official mirrors by default. You can customize the mirror source through the `TIUP_MIRRORS` environment variable.

TiUP mirrors can be a local file directory or an online HTTP server:

- `TIUP_MIRRORS=/path/to/local tiup list`
- `TIUP_MIRRORS=https://private-mirrors.example.com tiup list`

#### 10.4.4 Manage TiUP Components with TiUP Commands

You can use the following TiUP commands to manage components in the TiUP ecosystem:

- list: Queries the component list. By using this TiUP command, you can see all the optional components to install and all the optional versions of each component.
- install: Installs the specific version of a component.
- update: Updates a component to the latest version.
- uninstall: Uninstalls a component.
- status: Checks the status of a running component.
- clean: Cleans up the instance on which a component is deployed.
- help: Prints the help information. If you append another TiUP command to this command, the usage of the appended command is printed.

This document introduces the common component management operations and the corresponding TiUP commands.

##### 10.4.4.1 Query the component list

You can use the `tiup list` command to query the component list. This usage of this command is as follows:

- `tiup list`: checks which components can be installed.
- `tiup list ${component}`: checks which versions of a specific component can be installed.

You can also use the following flags in the above commands:

- `--installed`: checks which components or which version of a specific component has been installed locally. `--all`: views all components, including the hidden ones  
→ `verbose`: views all columns (including installed versions and supported platforms)

Example 1: View all currently installed components.

```
tiup list --installed
```

Example 2: Get a list of the TiKV component of all installable versions from the server.

```
tiup list tikv
```

#### 10.4.4.2 Install components

You can use the `tiup install` command to query the component list. This usage of this command is as follows:

- `tiup install <component>`: installs the latest stable version of a specified component.
- `tiup install <component>:[version]`: installs the specified version of a specified component.

Example 1: Use TiUP to install the latest stable version of TiDB.

```
tiup install tidb
```

Example 2: Use TiUP to install the nightly version of TiDB.

```
tiup install tidb:nightly
```

Example 3: Use TiUP to install TiKV v5.3.0.

```
tiup install tikv:v5.3.0
```

#### 10.4.4.3 Upgrade components

After a new version of a component is published, you can use the `tiup update` command to upgrade this component. The usage of this command is basically the same as that of `tiup → install`, except for the following flags:

- `--all`: Upgrades all components.
- `--nightly`: Upgrades to the nightly version.
- `--self`: Upgrades TiUP itself to the latest version.
- `--force`: Forcibly upgrades to the latest version.

Example 1: Upgrade all components to the latest versions.

```
tiup update --all
```

Example 2: Upgrade all components to the nightly version.

```
tiup update --all --nightly
```

Example 3: Upgrade TiUP to the latest version.

```
tiup update --self
```

#### 10.4.4.4 Operate components

After the installation is complete, you can use the `tiup <component>` command to start the corresponding component:

```
tiup [flags] <component>[:version] [args...]
```

Flags:

<code>-T, --tag string</code> $\hookrightarrow$ instance.	Specifies the tag for the component
--	-------------------------------------

To use this command, you need to specify the component name and the optional version. If no version is specified, the latest stable version (installed) of this component is used.

Before the component is started, TiUP creates a directory for it, and then puts this component into the directory for operation. The component generates all the data in this directory, and the name of this directory is the tag name specified when the component operates. If no tag is specified, a tag name is randomly generated. This working directory will be *automatically deleted* when the instance is terminated.

If you want to start the same component multiple times and reuse the previous working directory, you can use `--tag` to specify the same name when the component is started. After the tag is specified, the working directory will *not be automatically deleted* when the instance is terminated, which makes it convenient to reuse the working directory.

Example 1: Operate TiDB v5.3.0.

```
tiup tidb:v5.3.0
```

Example 2: Specify the tag with which TiKV operates.

```
tiup --tag=experiment tikv
```

#### 10.4.4.4.1 Query the operating status of a component

You can use the `tiup status` command to check the operating status of a component:

```
tiup status
```

By executing this command, you will get a list of instances, one instance per line. The list contains the following columns:

- **Name:** The tag name of the instance.
- **Component:** The component name of the instance.
- **PID:** The process ID of the operating instance.
- **Status:** The instance status. `RUNNING` means that the instance is operating. `TERM` means that the instance is terminated.
- **Created Time:** The starting time of the instance.

- **Directory:** The working directory of the instance, which can be specified using `--tag`.
- **Binary:** The executable program of the instance, which can be specified using `--binpath`.
- **Args:** The arguments of the operating instance.

#### 10.4.4.4.2 Clean component instance

You can use the `tiup clean` command to clean up component instances and delete the working directory. If the instance is still operating before the cleaning, the related process is killed first. The command usage is as follows:

```
tiup clean [tag] [flags]
```

The following flag is supported:

- `--all`: Cleans up all instance information.

In the above command, `tag` is the instance tag to be cleaned. If `--all` is used, no tag is passed.

Example 1: Clean up the component instance with the `experiment` tag name.

```
tiup clean experiment
```

Example 2: Clean up all component instances.

```
tiup clean --all
```

#### 10.4.4.4.3 Uninstall components

The components installed using TiUP take up local disk space. If you do not want to keep too many components of old versions, you can check which versions of a component are currently installed, and then uninstall this component.

You can use the `tiup uninstall` command to uninstall all versions or specific versions of a component. This command also supports uninstalling all components. The command usage is as follows:

```
tiup uninstall [component] [:version] [flags]
```

The following flags are supported in this command:

- `--all`: Uninstalls all components or versions.
- `--self`: Uninstalls TiUP itself.

`component` is the component to be uninstalled. `version` is the version to be uninstalled. Both `component` and `version` can be ignored in the `tiup uninstall` command. If you ignore either one of these two, you need to add the `--all` flag.

- If the version is ignored, adding `--all` means to uninstall all versions of this component.
- If the version and the component are both ignored, adding `--all` means to uninstall all components of all versions.

Example 1: Uninstall TiDB v5.3.0.

```
tiup uninstall tidb:v5.3.0
```

Example 2: Uninstall TiKV of all versions.

```
tiup uninstall tikv --all
```

Example 3: Uninstall all installed components.

```
tiup uninstall --all
```

## 10.4.5 TiUP FAQ

### 10.4.5.1 Can TiUP not use the official mirror source?

TiUP supports specifying the mirror source through the `TIUP_MIRRORS` environment variable. The address of the mirror source can be a local directory or an HTTP server address. If your environment cannot access the network, you can create your own offline mirror source to use TiUP.

After using an unofficial mirror, if you want the official mirror back and use it, take one of the following measures:

- Set the `TIUP_MIRRORS` variable to the official mirror address: `https://tiup-mirrors`  
↪ `.pingcap.com`.
- Make sure that the `TIUP_MIRRORS` variable is not set, and then execute the `tiup`  
↪ `mirror set https://tiup-mirrors.pingcap.com` command.

### 10.4.5.2 How do I put my own component into the TiUP mirrors?

TiUP does not support third-party components for the time being, but the TiUP Team has developed the TiUP component development specifications and is developing the `tiup-publish` component. After everything is ready, a contributor can publish their own components to TiUP's official mirrors by using the `tiup publish <comp> <version>` command.

### 10.4.5.3 What is the difference between the TiUP playground and TiUP cluster components?

The TiUP playground component is mainly used to build a stand-alone development environment on Linux or macOS operating systems. It helps you get started quickly and run a specified version of the TiUP cluster easily. The TiUP cluster component is mainly used to deploy and maintain a production environment cluster, which is usually a large-scale cluster.

#### 10.4.5.4 How do I write the topology file for the TiUP cluster component?

Refer to [these templates](#) to write the topology file. The templates include:

- Multi-DC deployment topology
- Minimal deployment topology
- Complete topology file

You can edit your topology file based on the templates and your needs.

#### 10.4.5.5 Can multiple instances be deployed on the same host?

You can use the TiUP cluster component to deploy multiple instances on the same host, but with different ports and directories configured; otherwise, directory and port conflicts might occur.

#### 10.4.5.6 Are port and directory conflicts detected within the same cluster?

Port and directory conflicts in the same cluster are detected during deployment and scaling. If there is any directory or port conflict, the deployment or scaling process is interrupted.

#### 10.4.5.7 Are port and directory conflicts detected among different clusters?

If multiple different clusters are deployed by the same TiUP control machine, the port and directory conflicts among these clusters are detected during deployment and scaling. If the clusters are deployed by different TiUP control machines, conflict detection is not supported currently.

#### 10.4.5.8 During cluster deployment, TiUP received an `ssh: handshake failed: read tcp 10.10.10.34:38980 -> 10.10.10.34:3600: read: connection reset by peer error`

The error might occur because the default number of concurrent threads of TiUP exceeds the default maximum number of SSH connections. To solve the issue, you can increase the default number of SSH connections, and then restart the sshd service:

```
vi /etc/ssh/sshd_config
```

```
MaxSessions 1000
MaxStartups 1000
```

### 10.4.6 TiUP Troubleshooting Guide

This document introduces some common issues when you use TiUP and the troubleshooting methods. If this document does not include the issues you bump into, [file a new issue](#) in the Github TiUP repository.

#### 10.4.6.1 Troubleshoot TiUP commands

##### 10.4.6.1.1 Can't see the latest component list using `tiup list`

TiUP does not update the latest component list from the mirror server every time. You can forcibly refresh the component list by running `tiup list`.

##### 10.4.6.1.2 Can't see the latest version information of a component using `tiup list <component>`

Same as the previous issue, the component version information is only obtained from the mirror server when there is no local cache. You can refresh the component list by running `tiup list <component>`.

##### 10.4.6.1.3 Component downloading process is interrupted

Unstable network might result in an interrupted component downloading process. You can try to download the component again. If you cannot download it after trying multiple times, it might be caused by the CDN server and you can report the issue [here](#).

##### 10.4.6.1.4 A checksum error occurs during component downloading process

Because the CDN server has a short cache time, the new checksum file might not match the component package. Try to download again after 5 minutes. If the new checksum file still does not match the component package, report the issue [here](#).

#### 10.4.6.2 Troubleshoot TiUP cluster component

##### 10.4.6.2.1 `unable to authenticate, attempted methods [none publickey]` is prompted during deployment

During deployment, component packages are uploaded to the remote host and the initialization is performed. This process requires connecting to the remote host. This error is caused by the failure to find the SSH private key to connect to the remote host.

To solve this issue, confirm whether you have specified the private key by running `tiup ↵ cluster deploy -i identity_file`:

- If the `-i` flag is not specified, it might be that TiUP does not automatically find the private key path. It is recommended to explicitly specify the private key path using `-i`.
- If the `-i` flag is specified, it might be that TiUP cannot log in to the remote host using the specified private key. You can verify it by manually executing the `ssh -i ↵ identity_file user@remote` command.
- If a password is used to log in to the remote host, make sure that you have specified the `-p` flag and entered the correct login password.

#### 10.4.6.2.2 The process of upgrading the cluster using the TiUP cluster component is interrupted

To avoid misuse cases, the TiUP cluster component does not support the upgrade of specified nodes, so after the upgrade fails, you need to perform the upgrade operations again, including idempotent operations during the upgrade process.

The upgrade process can be divided into the following steps:

1. Back up the old version of components on all nodes
2. Distribute new components to remote
3. Perform a rolling restart to all components

If the upgrade is interrupted during a rolling restart, instead of repeating the `tiup cluster upgrade` operation, you can use `tiup cluster restart -N <node1> -N <node2>` to restart the nodes that have not completed the restart.

If the number of un-restarted nodes of the same component is relatively large, you can also restart a certain type of component by running `tiup cluster restart -R <component>`.

#### 10.4.6.2.3 During the upgrade, you find that `node_exporter-9100.service/blackbox_exporter-9115.service` does not exist

If you previously migrated your cluster from TiDB Ansible and the exporter was not deployed in TiDB Ansible, this situation might happen. To solve it, you can manually copy the missing files from other nodes to the new node for the time being. The TiUP team will complete the missing components during the migration process.

### 10.4.7 TiUP Reference

TiUP serves as the package manager of the TiDB ecosystem. It manages components in the TiDB ecosystem, such as TiDB, PD, and TiKV.

#### 10.4.7.1 Syntax

```
tiup [flags] <command> [args...] # Executes a command
### or
tiup [flags] <component> [args...] # Runs a component
```

You can use the `help` command to get the information of a specific command. The summary of each command shows its parameters and their usage. Mandatory parameters are shown in angle brackets, and optional parameters are shown in square brackets.

`<command>` represents the command name. For the list of supported commands, see the [Command list](#) below. `<component>` represents the component name. For the list of supported components, see the [Component list](#) below.

### 10.4.7.2 Options

#### 10.4.7.2.1 -B, --binary

- If you enable this option, the specified binary file path is printed.
  - Executing `tiup -B/--binary <component>` will have the path of the latest stable installed `<component>` component printed. If `<component>` is not installed, an error is returned.
  - Executing `tiup -B/--binary <component>:<version>` will have the path of the installed `<component>` component's `<version>` printed. If this `<version>` is not printed, an error is returned.
- Data type: BOOLEAN
- This option is disabled by default and its default value is `false`. To enable this option, you can add this option to the command, and pass the `true` value or do not pass any value.

**Note:**

This option can only be used in commands of the `tiup [flags] <component> > [args...]` format.

#### 10.4.7.2.2 --binpath

**Note:**

This option can only be used in commands of the `tiup [flags] <component> > [args...]` format.

- Specifies the path of the component to be executed. When a component is executed, if you do not want to use the binary file in the TiUP mirror, you can add this option to specify using the binary file in a custom path.
- Data type: STRING

#### 10.4.7.2.3 --skip-version-check

**Note:**

This option is deprecated since v1.3.0.

- Skips the validity check for version numbers. By default, the specified version number can only be the semantic version.
- Data type: **BOOLEAN**
- This option is disabled by default and its default value is **false**. To enable this option, you can add this option to the command, and pass the **true** value or do not pass any value.

#### 10.4.7.2.4 -T, --tag

- Specifies a tag for the component to be started. Some components need to use disk storage during the execution, and TiUP allocates a temporary storage directory for this execution. If you want TiUP to allocate a fixed directory, you can use **-T**/**--tag** to specify the name of the directory, so that the same batch of files can be read and written in multiple executions with the same tag.
- Data type: **STRING**

#### 10.4.7.2.5 -v, --version

Prints the TiUP version.

#### 10.4.7.2.6 --help

Prints the help information.

### 10.4.7.3 Command list

TiUP has multiple commands, and these commands have multiple sub-commands. For the specific commands and their detailed descriptions, click the corresponding links in the list below:

- [install](#): Installs a component.
- [list](#): Shows the component list.
- [uninstall](#): Uninstalls a component.
- [update](#): Updates the installed component.
- [status](#): Shows the running status of a component.
- [clean](#): Cleans the data directory of a component.
- [mirror](#): Manages the mirror.

- [telemetry](#): Enables or disables the telemetry.
- [completion](#): Completes the TiUP command.
- [env](#): Shows the TiUP-related environment variables.
- [help](#): Shows the help information of a command or component.

#### 10.4.7.4 Component list

- [cluster](#): Manages the TiDB cluster in a production environment.
- [dm](#): Manages the TiDB Data Migration (DM) cluster in a production environment.

### 10.4.8 Topology Configuration File for TiDB Deployment Using TiUP

To deploy or scale TiDB using TiUP, you need to provide a topology file ([sample](#)) to describe the cluster topology.

Similarly, to modify the cluster topology, you need to modify the topology file. The difference is that, after the cluster is deployed, you can only modify a part of the fields in the topology file. This document introduces each section of the topology file and each field in each section.

#### 10.4.8.1 File structure

A topology configuration file for TiDB deployment using TiUP might contain the following sections:

- [global](#): The cluster's global configuration. Some of the configuration items use the default values and you can configure them separately in each instance.
- [monitored](#): Configuration for monitoring services, namely, the `blackbox_exporter` and the `node_exporter`. On each machine, a `node_exporter` and a `blackbox_exporter` are deployed.
- [server\\_configs](#): Components' global configuration. You can configure each component separately. If an instance has a configuration item with the same name, the instance's configuration item will take effect.
- [pd\\_servers](#): The configuration of the PD instance. This configuration specifies the machines to which the PD component is deployed.
- [tidb\\_servers](#): The configuration of the TiDB instance. This configuration specifies the machines to which the TiDB component is deployed.
- [tikv\\_servers](#): The configuration of the TiKV instance. This configuration specifies the machines to which the TiKV component is deployed.
- [tiflash\\_servers](#): The configuration of the TiFlash instance. This configuration specifies the machines to which the TiFlash component is deployed.
- [pump\\_servers](#): The configuration of the Pump instance. This configuration specifies the machines to which the Pump component is deployed.
- [drainer\\_servers](#): The configuration of the Drainer instance. This configuration specifies the machines to which the Drainer component is deployed.

- **cdc\_servers**: The configuration of the TiCDC instance. This configuration specifies the machines to which the TiCDC component is deployed.
- **tispark\_masters**: The configuration of the TiSpark master instance. This configuration specifies the machines to which the TiSpark master component is deployed. Only one node of TiSpark master can be deployed.
- **tispark\_workers**: The configuration of the TiSpark worker instance. This configuration specifies the machines to which the TiSpark worker component is deployed.
- **monitoring\_servers**: Specifies the machines to which Prometheus and NGMonitoring are deployed. TiUP supports deploying multiple Prometheus instances but only the first instance is used.
- **grafana\_servers**: The configuration of the Grafana instance. This configuration specifies the machines to which Grafana is deployed.
- **alertmanager\_servers**: The configuration of the Alertmanager instance. This configuration specifies the machines to which Alertmanager is deployed.

#### 10.4.8.1.1 global

The `global` section corresponds to the cluster's global configuration and has the following fields:

- **user**: The user used to start the deployed cluster. The default value is "`tidb`". If the user specified in the `<user>` field does not exist on the target machine, this user is automatically created.
- **group**: The user group to which a user belongs. It is specified when the user is created. The value defaults to that of the `<user>` field. If the specified group does not exist, it is automatically created.
- **ssh\_port**: Specifies the SSH port to connect to the target machine for operations. The default value is `22`.
- **enable\_tls**: Specifies whether to enable TLS for the cluster. After TLS is enabled, the generated TLS certificate must be used for connections between components or between the client and the component. **Once it is enabled, it cannot be disabled.** The default value is `false`.
- **deploy\_dir**: The deployment directory of each component. The default value is "`→ deployed`". Its application rules are as follows:
  - If the absolute path of `deploy_dir` is configured at the instance level, the actual deployment directory is `deploy_dir` configured for the instance.
  - For each instance, if you do not configure `deploy_dir`, its default value is the relative path `<component-name>-<component-port>`.
  - If `global.deploy_dir` is an absolute path, the component is deployed to the `<global.deploy_dir>/<instance.deploy_dir>` directory.

- If `global.deploy_dir` is a relative path, the component is deployed to the `/home/<global.user>/<global.deploy_dir>/<instance.deploy_dir>` directory.
- `data_dir`: The data directory. Default value: "data". Its application rules are as follows:
  - If the absolute path of `data_dir` is configured at the instance level, the actual deployment directory is `data_dir` configured for the instance.
  - For each instance, if you do not configure `data_dir`, its default value is `<global .data_dir>`.
  - If `data_dir` is a relative path, the component data is placed in `<deploy_dir>/<data_dir>`. For the calculation rules of `<deploy_dir>`, see the application rules of the `deploy_dir` field.
- `log_dir`: The log directory. Default value: "log". Its application rules are as follows:
  - If the absolute path `log_dir` is configured at the instance level, the actual log directory is the `log_dir` configured for the instance.
  - For each instance, if you not configure `log_dir`, its default value is `<global .log_dir>`.
  - If `log_dir` is a relative path, the component log is placed in `<deploy_dir>/<log_dir>`. For the calculation rules of `<deploy_dir>`, see the application rules of the `deploy_dir` field.
- `os`: The operating system of the target machine. The field controls which operating system to adapt to for the components pushed to the target machine. The default value is "linux".
- `arch`: The CPU architecture of the target machine. The field controls which platform to adapt to for the binary packages pushed to the target machine. The supported values are "amd64" and "arm64". The default value is "amd64".
- `resource_control`: Runtime resource control. All configurations in this field are written into the service file of systemd. There is no limit by default. The resources that can be controlled are as follows:
  - `memory_limit`: Limits the maximum runtime memory. For example, "2G" means that the maximum memory of 2 GB can be used.
  - `cpu_quota`: Limits the maximum CPU usage at runtime. For example, "200%".
  - `io_read_bandwidth_max`: Limits the maximum I/O bandwidth for disk reads. For example, "/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0 100M".
  - `io_write_bandwidth_max`: Limits maximum I/O bandwidth for disk writes. For example, /dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0 100M.

- `limit_core`: Controls the size of core dump.

A `global` configuration example is as follows:

```
global:
  user: "tidb"
  resource_control:
    memory_limit: "2G"
```

In the above configuration, the `tidb` user is used to start the cluster. At the same time, each component is restricted to a maximum of 2 GB of memory when it is running.

#### 10.4.8.1.2 `monitored`

`monitored` is used to configure the monitoring service on the target machine: `node_exporter` and `blackbox_exporter`. The following fields are included:

- `node_exporter_port`: The service port of `node_exporter`. The default value is 9100.
- `blackbox_exporter_port`: The service port of `blackbox_exporter`. The default value is 9115.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.

A `monitored` configuration example is as follows:

```
monitored:
  node_exporter_port: 9100
  blackbox_exporter_port: 9115
```

The above configuration specifies that `node_exporter` uses the 9100 port and `blackbox_exporter` uses the 9115 port.

#### 10.4.8.1.3 server\_configs

`server_configs` is used to configure services and to generate configuration files for each component. Similar to the `global` section, the configuration of this section can be overwritten by the configurations with the same names in an instance. `server_configs` mainly includes the following fields:

- `tidb`: TiDB service-related configuration. For the complete configuration, see [TiDB configuration file](#).
- `tikv`: TiKV service-related configuration. For the complete configuration, see [TiKV configuration file](#).
- `pd`: PD service-related configuration. For the complete configuration, see [PD configuration file](#).
- `tiflash`: TiFlash service-related configuration. For the complete configuration, see [TiFlash configuration file](#).
- `tiflash_learner`: Each TiFlash node has a special built-in TiKV. This configuration item is used to configure this special TiKV. It is generally not recommended to modify the content under this configuration item.
- `pump`: Pump service-related configuration. For the complete configuration, see [TiDB Binlog configuration file](#).
- `drainer`: Drainer service-related configuration. For the complete configuration, see [TiDB Binlog configuration file](#).
- `cdc`: TiCDC service-related configuration. For the complete configuration, see [Deploy TiCDC](#).

A `server_configs` configuration example is as follows:

```
server_configs:
  tidb:
    run-ddl: true
    lease: "45s"
    split-table: true
    token-limit: 1000
  tikv:
    log-level: "info"
    readpool.unified.min-thread-count: 1
```

The above configuration specifies the global configuration of TiDB and TiKV.

#### 10.4.8.1.4 pd\_servers

`pd_servers` specifies the machines to which PD services are deployed. It also specifies the service configuration on each machine. `pd_servers` is an array, and each element of the array contains the following fields:

- `host`: Specifies the machine to which the PD services are deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is `0.0.0.0`.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `name`: Specifies the name of the PD instance. Different instances must have unique names; otherwise, instances cannot be deployed.
- `client_port`: Specifies the port that PD uses to connect to the client. The default value is `2379`.
- `peer_port`: Specifies the port for communication between PDs. The default value is `2380`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as “`0,1`”.
- `config`: The configuration rule of this field is the same as the `pd` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `pd` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.

- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `name`
- `client_port`
- `peer_port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `pd_servers` configuration example is as follows:

```
pd_servers:
- host: 10.0.1.11
  config:
    schedule.max-merge-region-size: 20
    schedule.max-merge-region-keys: 200000
- host: 10.0.1.12
```

The above configuration specifies that PD will be deployed on 10.0.1.11 and 10.0.1.12, and makes specific configurations for the PD of 10.0.1.11.

#### 10.4.8.1.5 `tidb_servers`

`tidb_servers` specifies the machines to which TiDB services are deployed. It also specifies the service configuration on each machine. `tidb_servers` is an array, and each element of the array contains the following fields:

- `host`: Specifies the machine to which the TiDB services are deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is 0.0.0.0.

- **ssh\_port**: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- **port**: The listening port of TiDB services, which is used to provide connection to the MySQL client. The default value is 4000.
- **status\_port**: The listening port of the TiDB status service, which is used to view the status of the TiDB services from the external via HTTP requests. The default value is 10080.
- **deploy\_dir**: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- **log\_dir**: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- **numa\_node**: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as “0,1”.
- **config**: The configuration rule of this field is the same as the `tidb` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `tidb` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- **os**: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `port`
- `status_port`
- `deploy_dir`

- `log_dir`
- `arch`
- `os`

A `tidb_servers` configuration example is as follows:

```
tidb_servers:
  - host: 10.0.1.14
    config:
      log.level: warn
      log.slow-query-file: tidb-slow-overwrited.log
  - host: 10.0.1.15
```

#### 10.4.8.1.6 `tikv_servers`

`tikv_servers` specifies the machines to which TiKV services are deployed. It also specifies the service configuration on each machine. `tikv_servers` is an array, and each element of the array contains the following fields:

- `host`: Specifies the machine to which the TiKV services are deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is `0.0.0.0`.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of the TiKV services. The default value is `20160`.
- `status_port`: The listening port of the TiKV status service. The default value is `20180`.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as “`0,1`”.

- **config**: The configuration rule of this field is the same as the `tikv` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `tikv` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- **os**: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `port`
- `status_port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `tikv_servers` configuration example is as follows:

```
tikv_servers:
- host: 10.0.1.14
  config:
    server.labels: { zone: "zone1", host: "host1" }
- host: 10.0.1.15
  config:
    server.labels: { zone: "zone1", host: "host2" }
```

#### 10.4.8.1.7 `tiflash_servers`

`tiflash_servers` specifies the machines to which TiFlash services are deployed. It also specifies the service configuration on each machine. This section is an array, and each element of the array contains the following fields:

- **host**: Specifies the machine to which the TiFlash services are deployed. The field value is an IP address and is mandatory.
- **ssh\_port**: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- **tcp\_port**: The port of the TiFlash TCP service. The default value is 9000.
- **http\_port**: The port of the TiFlash HTTP service. The default value is 8123.
- **flash\_service\_port**: The port via which TiFlash provides services. TiDB reads data from TiFlash via this port. The default value is 3930.
- **metrics\_port**: TiFlash's status port, which is used to output metric data. The default value is 8234.
- **flash\_proxy\_port**: The port of the built-in TiKV. The default value is 20170.
- **flash\_proxy\_status\_port**: The status port of the built-in TiKV. The default value is 20292.
- **deploy\_dir**: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- **data\_dir**: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`. TiFlash supports multiple `data_dir` directories separated by commas.
- **log\_dir**: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- **tmp\_path**: The storage path of TiFlash temporary files. The default value is [path or the first directory of `storage.latest.dir`] + “/tmp”.
- **numa\_node**: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as “0,1”.
- **config**: The configuration rule of this field is the same as the `tiflash` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `tiflash` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- **learner\_config**: Each TiFlash node has a special built-in TiKV. This configuration item is used to configure this special TiKV. It is generally not recommended to modify the content under this configuration item.

- **os**: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

After the deployment, for the fields above, you can only add directories to `data_dir`; for the fields below, you cannot modified these fields:

- `host`
- `tcp_port`
- `http_port`
- `flash_service_port`
- `flash_proxy_port`
- `flash_proxy_status_port`
- `metrics_port`
- `deploy_dir`
- `log_dir`
- `tmp_path`
- `arch`
- `os`

A `tiflash_servers` configuration example is as follows:

```
tiflash_servers:
  - host: 10.0.1.21
  - host: 10.0.1.22
```

#### 10.4.8.1.8 `pump_servers`

`pump_servers` specifies the machines to which the Pump services of TiDB Binlog are deployed. It also specifies the service configuration on each machine. `pump_servers` is an array, and each element of the array contains the following fields:

- **host**: Specifies the machine to which the Pump services are deployed. The field value is an IP address and is mandatory.
- **ssh\_port**: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.

- **port**: The listening port of the Pump services. The default value is 8250.
- **deploy\_dir**: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- **data\_dir**: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- **log\_dir**: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- **numa\_node**: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as “0,1”.
- **config**: The configuration rule of this field is the same as the `pump` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `pump` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.
- **os**: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `pump_servers` configuration example is as follows:

```
pump_servers:
  - host: 10.0.1.21
    config:
      gc: 7
  - host: 10.0.1.22
```

#### 10.4.8.1.9 drainer\_servers

`drainer_servers` specifies the machines to which the Drainer services of TiDB Binlog are deployed. It also specifies the service configuration on each machine. `drainer_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the Drainer services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of Drainer services. The default value is 8249.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `commit_ts`: When Drainer starts, it reads the checkpoint. If Drainer cannot read the checkpoint, it uses this field as the replication time point for the initial startup. This field defaults to -1 (Drainer always gets the latest timestamp from the PD as the `commit_ts`).
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- `config`: The configuration rule of this field is the same as the `drainer` configuration rule in `server_configs`. If this field is configured, the field content is merged with the `drainer` content in `server_configs` (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in `host`.

- **os**: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `commit_ts`
- `arch`
- `os`

A `drainer_servers` configuration example is as follows:

```
drainer_servers:
  - host: 10.0.1.21
    config:
      syncer.db-type: "mysql"
      syncer.to.host: "127.0.0.1"
      syncer.to.user: "root"
      syncer.to.password: ""
      syncer.to.port: 3306
      syncer.ignore-table:
        - db-name: test
          tbl-name: log
        - db-name: test
          tbl-name: audit
```

#### 10.4.8.1.10 `cdc_servers`

`cdc_servers` specifies the machines to which the TiCDC services are deployed. It also specifies the service configuration on each machine. `cdc_servers` is an array. Each array element contains the following fields:

- **host**: Specifies the machine to which the TiCDC services are deployed. The field value is an IP address and is mandatory.
- **ssh\_port**: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the **ssh\_port** of the **global** section is used.
- **port**: The listening port of the TiCDC services. The default value is 8300.
- **deploy\_dir**: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the **deploy\_dir** directory configured in **global**.
- **data\_dir**: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the **data\_dir** directory configured in **global**.
- **log\_dir**: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the **log\_dir** directory configured in **global**.
- **gc-ttl**: The Time To Live (TTL) duration of the service level GC safepoint set by TiCDC in PD, in seconds. The default value is 86400, which is 24 hours.
- **tz**: The time zone that the TiCDC services use. TiCDC uses this time zone when internally converting time data types such as timestamp and when replicating data to the downstream. The default value is the local time zone where the process runs.
- **numa\_node**: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has **numactl** installed. If this field is specified, cpubind and membind policies are allocated using **numactl**. This field is the string type. The field value is the ID of the NUMA node, such as “0,1”.
- **config**: The field content is merged with the **cdc** content in **server\_configs** (if the two fields overlap, the content of this field takes effect). Then, a configuration file is generated and sent to the machine specified in **host**.
- **os**: The operating system of the machine specified in **host**. If this field is not specified, the default value is the **os** value in **global**.
- **arch**: The architecture of the machine specified in **host**. If this field is not specified, the default value is the **arch** value in **global**.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the **resource\_control** content in **global** (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in **host**. The configuration rules of **resource\_control** are the same as the **resource\_control** content in **global**.

For the above fields, you cannot modify these configured fields after the deployment:

- host
- port
- deploy\_dir
- data\_dir
- log\_dir
- arch
- os

A `cdc_servers` configuration example is as follows:

```
cdc_servers:
  - host: 10.0.1.20
    gc-ttl: 86400
    data_dir: "/cdc-data"
  - host: 10.0.1.21
    gc-ttl: 86400
    data_dir: "/cdc-data"
```

#### 10.4.8.1.11 tispark\_masters

`tispark_masters` specifies the machines to which the master node of TiSpark is deployed. It also specifies the service configuration on each machine. `tispark_masters` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the TiSpark master is deployed. The field value is an IP address and is mandatory.
- `listen_host`: When the machine has multiple IP addresses, `listen_host` specifies the listening IP address of the service. The default value is 0.0.0.0.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: Spark's listening port, used for communication before the node. The default value is 7077.
- `web_port`: Spark's web port, which provides web services and the task status. The default value is 8080.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `java_home`: Specifies the path of the JRE environment to be used. This parameter corresponds to the `JAVA_HOME` system environment variable.

- **spark\_config**: Configures to configure the TiSpark services. Then, a configuration file is generated and sent to the machine specified in `host`.
- **spark\_env**: Configures the environment variables when Spark starts.
- **os**: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `listen_host`
- `port`
- `web_port`
- `deploy_dir`
- `arch`
- `os`

A `tispark_masters` configuration example is as follows:

```
tispark_masters:
- host: 10.0.1.21
  spark_config:
    spark.driver.memory: "2g"
    spark.eventLog.enabled: "False"
    spark.tispark.grpc.framesize: 2147483647
    spark.tispark.grpc.timeout_in_sec: 100
    spark.tispark.meta.reload_period_in_sec: 60
    spark.tispark.request.command.priority: "Low"
    spark.tispark.table.scan_concurrency: 256
  spark_env:
    SPARK_EXECUTOR_CORES: 5
    SPARK_EXECUTOR_MEMORY: "10g"
    SPARK_WORKER_CORES: 5
    SPARK_WORKER_MEMORY: "10g"
- host: 10.0.1.22
```

#### 10.4.8.1.12 tispark\_workers

`tispark_workers` specifies the machines to which the worker nodes of TiSpark are deployed. It also specifies the service configuration on each machine. `tispark_workers` is an array. Each array element contains the following fields:

- **host**: Specifies the machine to which the TiSpark workers are deployed. The field value is an IP address and is mandatory.
- **listen\_host**: When the machine has multiple IP addresses, **listen\_host** specifies the listening IP address of the service. The default value is 0.0.0.0.
- **ssh\_port**: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the **ssh\_port** of the **global** section is used.
- **port**: Spark's listening port, used for communication before the node. The default value is 7077.
- **web\_port**: Spark's web port, which provides web services and the task status. The default value is 8080.
- **deploy\_dir**: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the **deploy\_dir** directory configured in **global**.
- **java\_home**: Specifies the path in which the JRE environment to be used is located. This parameter corresponds to the **JAVA\_HOME** system environment variable.
- **os**: The operating system of the machine specified in **host**. If this field is not specified, the default value is the **os** value in **global**.
- **arch**: The architecture of the machine specified in **host**. If this field is not specified, the default value is the **arch** value in **global**.

For the above fields, you cannot modify these configured fields after the deployment:

- **host**
- **listen\_host**
- **port**
- **web\_port**
- **deploy\_dir**
- **arch**
- **os**

A **tispark\_workers** configuration example is as follows:

```
tispark_workers:
  - host: 10.0.1.22
  - host: 10.0.1.23
```

#### 10.4.8.1.13 monitoring\_servers

`monitoring_servers` specifies the machines to which the Prometheus services are deployed. It also specifies the service configuration on each machine. `monitoring_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the monitoring services are deployed. The field value is an IP address and is mandatory.
- `ng_port`: Specifies the SSH port connecting to NGMonitoring. Introduced in TiUP v1.7.0, this field supports [Continuous Profiling](#) and Top SQL in TiDB 5.3.0 and above.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of the Prometheus services. The default value is 9090.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as "0,1".
- `storage_retention`: The retention time of the Prometheus monitoring data. The default value is "15d".
- `rule_dir`: Specifies a local directory that should contain complete `*.rules.yml` files. These files are transferred to the target machine during the initialization phase of the cluster configuration as the rules for Prometheus.
- `remote_config`: Supports writing Prometheus data to the remote, or reading data from the remote. This field has two configurations:
  - `remote_write`: See the Prometheus document [`<remote\_write>`](#).
  - `remote_read`: See the Prometheus document [`<remote\_read>`](#).
- `external_alertmanagers`: If the `external_alertmanagers` field is configured, Prometheus alerts the configuration behavior to the Alertmanager that is outside the cluster. This field is an array, each element of which is an external Alertmanager and consists of the `host` and `web_port` fields.

- **os**: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- **arch**: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

For the above fields, you cannot modify these configured fields after the deployment:

- `host`
- `port`
- `deploy_dir`
- `data_dir`
- `log_dir`
- `arch`
- `os`

A `monitoring_servers` configuration example is as follows:

```
monitoring_servers:
  - host: 10.0.1.11
    rule_dir: /local/rule/dir
    remote_config:
      remote_write:
        - queue_config:
            batch_send_deadline: 5m
            capacity: 100000
            max_samples_per_send: 10000
            max_shards: 300
            url: http://127.0.0.1:8003/write
      remote_read:
        - url: http://127.0.0.1:8003/read
    external_alertmanagers:
      - host: 10.1.1.1
        web_port: 9093
      - host: 10.1.1.2
        web_port: 9094
```

#### 10.4.8.1.14 `grafana_servers`

`grafana_servers` specifies the machines to which the Grafana services are deployed. It also specifies the service configuration on each machine. `grafana_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the Grafana services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `port`: The listening port of the Grafana services. The default value is 3000.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `os`: The operating system of the machine specified in `host`. If this field is not specified, the default value is the `os` value in `global`.
- `arch`: The architecture of the machine specified in `host`. If this field is not specified, the default value is the `arch` value in `global`.
- `username`: The user name on the Grafana login interface.
- `password`: The password corresponding to Grafana.
- `dashboard_dir`: Specifies a local directory that should contain complete `dashboard` ↴ (`*.json`) files. These files are transferred to the target machine during the initialization phase of the cluster configuration as the dashboards for Grafana.
- `resource_control`: Resource control for the service. If this field is configured, the field content is merged with the `resource_control` content in `global` (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in `host`. The configuration rules of `resource_control` are the same as the `resource_control` content in `global`.

##### Note:

If the `dashboard_dir` field of `grafana_servers` is configured, after executing the `tiup cluster rename` command to rename the cluster, you need to perform the following operations:

1. For the `*.json` files in the local dashboards directory, update the value of the `datasource` field to the new cluster name (because `datasource` is named after the cluster name).
2. Execute the `tiup cluster reload -R grafana` command.

For the above fields, you cannot modify these configured fields after the deployment:

- host
- port
- deploy\_dir
- arch
- os

A `grafana_servers` configuration example is as follows:

```
grafana_servers:
  - host: 10.0.1.11
    dashboard_dir: /local/dashboard/dir
```

#### 10.4.8.1.15 alertmanager\_servers

`alertmanager_servers` specifies the machines to which the Alertmanager services are deployed. It also specifies the service configuration on each machine. `alertmanager_servers` is an array. Each array element contains the following fields:

- `host`: Specifies the machine to which the Alertmanager services are deployed. The field value is an IP address and is mandatory.
- `ssh_port`: Specifies the SSH port to connect to the target machine for operations. If it is not specified, the `ssh_port` of the `global` section is used.
- `web_port`: Specifies the port used that Alertmanager uses to provide web services. The default value is 9093.
- `cluster_port`: Specifies the communication port between one Alertmanger and other Alertmanager. The default value is 9094.
- `deploy_dir`: Specifies the deployment directory. If it is not specified or specified as a relative directory, the directory is generated according to the `deploy_dir` directory configured in `global`.
- `data_dir`: Specifies the data directory. If it is not specified or specified as a relative directory, the directory is generated according to the `data_dir` directory configured in `global`.
- `log_dir`: Specifies the log directory. If it is not specified or specified as a relative directory, the log is generated according to the `log_dir` directory configured in `global`.
- `numa_node`: Allocates the NUMA policy to the instance. Before specifying this field, you need to make sure that the target machine has `numactl` installed. If this field is specified, cpubind and membind policies are allocated using `numactl`. This field is the string type. The field value is the ID of the NUMA node, such as “0,1”.

- **config\_file**: Specifies a local file that is transferred to the target machine during the initialization phase of the cluster configuration as the configuration of Alertmanager.
- **os**: The operating system of the machine specified in **host**. If this field is not specified, the default value is the **os** value in **global**.
- **arch**: The architecture of the machine specified in **host**. If this field is not specified, the default value is the **arch** value in **global**.
- **resource\_control**: Resource control for the service. If this field is configured, the field content is merged with the **resource\_control** content in **global** (if the two fields overlap, the content of this field takes effect). Then, a systemd configuration file is generated and sent to the machine specified in **host**. The configuration rules of **resource\_control** are the same as the **resource\_control** content in **global**.

For the above fields, you cannot modify these configured fields after the deployment:

- **host**
- **web\_port**
- **cluster\_port**
- **deploy\_dir**
- **data\_dir**
- **log\_dir**
- **arch**
- **os**

A `alertmanager_servers` configuration example is as follows:

```
alertmanager_servers:
  - host: 10.0.1.11
    config_file: /local/config/file
  - host: 10.0.1.12
    config_file: /local/config/file
```

#### 10.4.9 TiUP Mirror Reference Guide

TiUP mirrors are TiUP's component warehouse, which stores components and their metadata. TiUP mirrors take the following two forms:

- Directory on the local disk: serves the local TiUP client, which is called a local mirror in this document.
- HTTP mirror started based on the remote disk directory: serves the remote TiUP client, which is called a remote mirror in this document.

#### 10.4.9.1 Create and update mirror

You can create a TiUP mirror using one of the following two methods:

- Execute `tiup mirror init` to create a mirror from scratch.
- Execute `tiup mirror clone` to clone from an existing mirror.

After the mirror is created, you can add components to or delete components from the mirror using the `tiup mirror` commands. TiUP updates a mirror by adding files and assigning a new version number to it, rather than deleting any files from the mirror.

#### 10.4.9.2 Mirror structure

A typical mirror structure is as follows:

```
+ <mirror-dir>                                # Mirror's root directory
|-- root.json                                    # Mirror's root certificate
|-- {2..N}.root.json                            # Mirror's root certificate
|-- {1..N}.index.json                           # Component/user index
|-- {1..N}.{component}.json                     # Component metadata
|-- {component}-{version}-{os}-{arch}.tar.gz    # Component binary package
|-- snapshot.json                               # Mirror's latest snapshot
|-- timestamp.json                             # Mirror's latest timestamp
|--- commits                                     # Mirror's update log (deletable)
|   |-- commit-{ts1..tsN}
|       |-- {N}.root.json
|       |-- {N}.{component}.json
|       |-- {N}.index.json
|       |-- {component}-{version}-{os}-{arch}.tar.gz
|       |-- snapshot.json
|       |-- timestamp.json
|--- keys                                         # Mirror's private key (can be
|     ↪ moved to other locations)
|--- {hash1..hashN}-root.json                   # Private key of the root
|     ↪ certificate
|--- {hash}-index.json                          # Private key of the indexes
|--- {hash}-snapshot.json                      # Private key of the snapshots
|--- {hash}-timestamp.json                     # Private key of the timestamps
```

#### Note:

- The `commits` directory stores the logs generated in the process of mirror update and is used to roll back the mirror. You can delete the old log directories regularly when the disk space is insufficient.

- The private key stored in the `keys` directory is sensitive. It is recommended to keep it separately.

#### 10.4.9.2.1 Root directory

In a TiUP mirror, the root certificate is used to store the public key of other metadata files. Each time any metadata file (`*.json`) is obtained, TiUP client needs to find the corresponding public key in the installed `root.json` based on the metadata file type (root, index, snapshot, timestamp). Then TiUP client uses the public key to verify whether the signature is valid.

The root certificate's format is as follows:

```
{
  "signatures": [                                     # Each metadata file
    ↪ has some signatures which are signed by several private keys
    ↪ corresponding to the file.
    {
      "keyid": "{id-of-root-key-1}",                 # The ID of the
      ↪ first private key that participates in the signature. This
      ↪ ID is obtained by hashing the content of the public key
      ↪ that corresponds to the private key.
      "sig": "{signature-by-root-key-1}"             # The signed part of
      ↪ this file by this private key.
    },
    ...
    {
      "keyid": "{id-of-root-key-N}",                 # The ID of the Nth
      ↪ private key that participates in the signature.
      "sig": "{signature-by-root-key-N}"             # The signed part of
      ↪ this file by this private key.
    }
  ],
  "signed": {                                       # The signed part.
    "_type": "root",                                # The type of this
    ↪ file. root.json's type is root.
    "expires": "{expiration-date-of-this-file}",     # The expiration
    ↪ time of the file. If the file expires, the client rejects the
    ↪ file.
    "roles": {                                      # Records the keys
      ↪ used to sign each metadata file.
      "{role:index,root,snapshot,timestamp}": {      # Each involved
        ↪ metadata file includes index, root, snapshot, and timestamp
        ↪ .
      }
    }
  }
}
```

```

"keys": {                                     # Only the key's
    ↪ signature recorded in `keys` is valid.
    "{id-of-the-key-1)": {                   # The ID of the
        ↪ first key used to sign {role}.
        "keytype": "rsa",                  # The key's type.
        ↪ Currently, the key type is fixed as rsa.
        "keyval": {                      # The key's payload.
            "public": "{public-key-content}" # The public key's
                ↪ content.
        },
        "scheme": "rsassa-pss-sha256"      # Currently, the
            ↪ scheme is fixed as rsassa-pss-sha256.
    },
    "{id-of-the-key-N)": {                   # The ID of the Nth
        ↪ key used to sign {role}.
        "keytype": "rsa",
        "keyval": {
            "public": "{public-key-content}"
        },
        "scheme": "rsassa-pss-sha256"
    }
},
"threshold": {N},                           # Indicates that the
    ↪ metadata file needs at least N key signatures.
"url": "/{role}.json"                      # The address from
    ↪ which the file can be obtained. For index files, prefix
    ↪ it with the version number (for example, /{N}.index.
    ↪ json).
}
},
"spec_version": "0.1.0",                   # The specified
    ↪ version followed by this file. If the file structure is
    ↪ changed in the future, the version number needs to be upgraded
    ↪ . The current version number is 0.1.0.
"version": {N}                            # The version number
    ↪ of this file. You need to create a new {N+1}.root.json every
    ↪ time you update the file, and set its version to N + 1.
}
}

```

#### 10.4.9.2.2 Index

The index file records all the components in the mirror and the owner information of the components.

The index file's format is as follows:

```
{
  "signatures": [                                     # The file's
    ↪ signature.
    {
      "keyid": "{id-of-index-key-1}",                 # The ID of the
      ↪ first private key that participates in the signature.
      "sig": "{signature-by-index-key-1}",            # The signed part of
      ↪ this file by this private key.
    },
    ...
    {
      "keyid": "{id-of-root-key-N}",                  # The ID of the Nth
      ↪ private key that participates in the signature.
      "sig": "{signature-by-root-key-N}"              # The signed part of
      ↪ this file by this private key.
    }
  ],
  "signed": {
    "_type": "index",                                # The file type.
    "components": {                                    # The component list
      ↪ .
      "{component1}": {                               # The name of the
        ↪ first component.
        "hidden": {bool},                            # Whether it is a
        ↪ hidden component.
        "owner": "{owner-id}",                      # The component
        ↪ owner's ID.
        "standalone": {bool},                       # Whether it is a
        ↪ standalone component.
        "url": "/{component}.json",                 # The address from
        ↪ which the component can be obtained. You need to prefix
        ↪ it with the version number (for example, /{N}.{
        ↪ component}.json).
        "yanked": {bool}                           # Indicates whether
        ↪ the component is marked as deleted.
      },
      ...
      "{componentN}": {                             # The name of the
        ↪ Nth component.
        ...
      },
      "default_components": ["{component1}"..."{componentN}"], # The default
    }
  }
}
```

```

    ↵ component that a mirror must contain. Currently, this field
    ↵ defaults to empty (disabled).
"expires": "{expiration-date-of-this-file}",      # The expiration
    ↵ time of the file. If the file expires, the client rejects the
    ↵ file.
"owners": {
    "{owner1}": {                                # The ID of the
        ↵ first owner.
        "keys": {                                # Only the key's
            ↵ signature recorded in `keys` is valid.
            "{id-of-the-key-1)": {                # The first key of
                ↵ the owner.
                "keytype": "rsa",                  # The key's type.
                    ↵ Currently, the key type is fixed as rsa.
                "keyval": {                      # The key's payload.
                    "public": "{public-key-content}" # The public key's
                        ↵ content.
                },
                "scheme": "rsassa-pss-sha256"     # Currently, the
                    ↵ scheme is fixed as rsassa-pss-sha256.
            },
            ...
            "{id-of-the-key-N)": {              # The Nth key of the
                ↵ owner.
                ...
            }
        },
        "name": "{owner-name}",                  # The name of the
            ↵ owner.
        "threshod": {N}                      # Indicates that the
            ↵ components owned by the owner must have at least N
            ↵ valid signatures.
    },
    ...
    "{ownerN)": {                            # The ID of the Nth
        ↵ owner.
        ...
    }
}
"spec_version": "0.1.0",                  # The specified
    ↵ version followed by this file. If the file structure is
    ↵ changed in the future, the version number needs to be upgraded
    ↵ . The current version number is 0.1.0.
"version": {N}                          # The version number
    ↵ of this file. You need to create a new {N+1}.index.json every

```

```

    }                                ↪ time you update the file, and set its version to N + 1.
}
}
```

#### 10.4.9.2.3 Component

The component's metadata file records information of the component-specific platform and the version.

The component metadata file's format is as follows:

```
{
  "signatures": [                                # The file's
    → signature.
    {
      "keyid": "{id-of-index-key-1}",            # The ID of the
          → first private key that participates in the signature.
      "sig": "{signature-by-index-key-1}",        # The signed part of
          → this file by this private key.
    },
    ...
    {
      "keyid": "{id-of-root-key-N}",             # The ID of the Nth
          → private key that participates in the signature.
      "sig": "{signature-by-root-key-N}"         # The signed part of
          → this file by this private key.
    }
  ],
  "signed": {
    "_type": "component",                      # The file type.
    "description": "{description-of-the-component}", # The description of
        → the component.
    "expires": "{expiration-date-of-this-file}",   # The expiration
        → time of the file. If the file expires, the client rejects the
        → file.
    "id": "{component-id}",                     # The globally
        → unique ID of the component.
    "nightly": "{nightly-cursor}",              # The nightly cursor
        → , and the value is the latest nightly version number (for
        → example, v5.0.0-nightly-20201209).
    "platforms": {                                # The component's
      → supported platforms (such as darwin/amd64, linux/arm64).
      "{platform-pair-1}": {
        "{version-1)": {                         # The semantic
          → version number (for example, v1.0.0).
        }
      }
    }
  }
}
```

```

    "dependencies": null,                                # Specifies the
        ↪ dependency relationship between components. The
        ↪ field is not used yet and is fixed as null.
    "entry": "{entry}",                                 # The relative path
        ↪ of the entry binary file in the tar package.
    "hashes": {                                         # The checksum of
        ↪ the tar package. sha256 and sha512 are used.
        "sha256": "{sum-of-sha256}",
        "sha512": "{sum-of-sha512}",
    },
    "length": {length-of-tar},                           # The length of the
        ↪ tar package.
    "released": "{release-time}",                      # The release date
        ↪ of the version.
    "url": "{url-of-tar}",                            # The download
        ↪ address of the tar package.
    "yanked": {bool}                                    # Indicates whether
        ↪ this version is disabled.
}
},
...
"{platform-pair-N)": {
    ...
},
"spec_version": "0.1.0",                            # The specified
    ↪ version followed by this file. If the file structure is
    ↪ changed in the future, the version number needs to be upgraded
    ↪ . The current version number is 0.1.0.
"version": {N}                                     # The version number
    ↪ of this file. You need to create a new {N+1}.{component}.json
    ↪ every time you update the file, and set its version to N + 1.
}

```

#### 10.4.9.2.4 Snapshot

The snapshot file records the version number of each metadata file:

The snapshot file's structure is as follows:

```
{
    "signatures": [                                # The file's
        ↪ signature.
    {
        "keyid": "{id-of-index-key-1}",            # The ID of the
        ↪ first private key that participates in the signature.
    }
}
```

```

    "sig": "{signature-by-index-key-1}",           # The signed part of
        ↪ this file by this private key.
},
...
{
    "keyid": "{id-of-root-key-N}",                 # The ID of the Nth
        ↪ private key that participates in the signature.
    "sig": "{signature-by-root-key-N}"             # The signed part of
        ↪ this file by this private key.
}
],
"signed": {
    "_type": "snapshot",                         # The file type.
    "expires": "{expiration-date-of-this-file}",   # The expiration
        ↪ time of the file. If the file expires, the client rejects the
        ↪ file.
    "meta": {                                     # Other metadata
        ↪ files' information.
        "/root.json": {
            "length": {length-of-json-file},       # The length of root
                ↪ .json
            "version": {version-of-json-file}      # The version of
                ↪ root.json
        },
        "/index.json": {
            "length": {length-of-json-file},
            "version": {version-of-json-file}
        },
        "/{component-1}.json": {
            "length": {length-of-json-file},
            "version": {version-of-json-file}
        },
        ...
        "/{component-N}.json": {
            ...
        }
    },
    "spec_version": "0.1.0",                      # The specified
        ↪ version followed by this file. If the file structure is
        ↪ changed in the future, the version number needs to be upgraded
        ↪ . The current version number is 0.1.0.
    "version": 0                                # The version number
        ↪ of this file, which is fixed as 0.
}

```

#### 10.4.9.2.5 Timestamp

The timestamp file records the checksum of the current snapshot.

The timestamp file's format is as follows:

```
{
  "signatures": [                                     # The file's
    ↪ signature.
    {
      "keyid": "{id-of-index-key-1}",                 # The ID of the
      ↪ first private key that participates in the signature.
      "sig": "{signature-by-index-key-1}",            # The signed part of
      ↪ this file by this private key.
    },
    ...
    {
      "keyid": "{id-of-root-key-N}",                  # The ID of the Nth
      ↪ private key that participates in the signature.
      "sig": "{signature-by-root-key-N}"              # The signed part of
      ↪ this file by this private key.
    }
  ],
  "signed": {
    "_type": "timestamp",                           # The file type.
    "expires": "{expiration-date-of-this-file}",   # The expiration
    ↪ time of the file. If the file expires, the client rejects the
    ↪ file.
    "meta": {                                       # The information of
      ↪ snapshot.json.
      "/snapshot.json": {
        "hashes": {
          "sha256": "{sum-of-sha256}"                # snapshot.json's
          ↪ sha256.
        },
        "length": {length-of-json-file}             # The length of
          ↪ snapshot.json.
      }
    },
    "spec_version": "0.1.0",                      # The specified
    ↪ version followed by this file. If the file structure is
    ↪ changed in the future, the version number needs to be upgraded
    ↪ . The current version number is 0.1.0.
    "version": {N}                                # The version number
    ↪ of this file. You need to overwrite timestamp.json every time
    ↪ you update the file, and set its version to N + 1.
  }
}
```

### 10.4.9.3 Client workflow

The client uses the following logic to ensure that the files downloaded from the mirror are safe:

- A `root.json` file is included with the binary when the client is installed.
- The running client performs the following tasks based on the existing `root.json`:
  1. Obtain the version from `root.json` and mark it as N.
  2. Request `{N+1}.root.json` from the mirror. If the request is successful, use the public key recorded in `root.json` to verify whether the file is valid.
  3. Request `timestamp.json` from the mirror and use the public key recorded in `root.json` to verify whether the file is valid.
  4. Check whether the checksum of `snapshot.json` recorded in `timestamp.json` matches the checksum of the local `snapshot.json`. If the two do not match, request the latest `snapshot.json` from the mirror and use the public key recorded in `root.json` to verify whether the file is valid.
  5. Obtain the version number N of the `index.json` file from `snapshot.json` and request `{N}.index.json` from the mirror. Then use the public key recorded in `root.json` to verify whether the file is valid.
  6. For components such as `tidb.json` and `tikv.json`, the client obtains the version numbers N of the components from `snapshot.json` and requests `{N}.{component}→.json` from the mirror. Then the client uses the public key recorded in `index.json` to verify whether the file is valid.
  7. For component's tar files, the client obtains the URLs and checksums of the files from `{component}.json` and request the URLs for the tar packages. Then the client verifies whether the checksum is correct.

## 10.4.10 TiUP Components

### 10.4.10.1 Quickly Deploy a Local TiDB Cluster

The TiDB cluster is a distributed system that consists of multiple components. A typical TiDB cluster consists of at least three PD nodes, three TiKV nodes, and two TiDB nodes. If you want to have a quick experience on TiDB, you might find it time-consuming and complicated to manually deploy so many components. This document introduces the playground component of TiUP and how to use it to quickly build a local TiDB test environment.

#### 10.4.10.1.1 TiUP playground overview

The basic usage of the playground component is shown as follows:

```
tiup playground [version] [flags]
```

If you directly execute the `tiup playground` command, TiUP uses the locally installed TiDB, TiKV, and PD components or installs the stable version of these components to start

a TiDB cluster that consists of one TiKV instance, one TiDB instance, one PD instance, and one TiFlash instance.

This command actually performs the following operations:

- Because this command does not specify the version of the playground component, TiUP first checks the latest version of the installed playground component. Assume that the latest version is v1.7.0, then this command works the same as `tiup playground:v1 ↳ .7.0`.
- If you have not used TiUP playground to install the TiDB, TiKV, and PD components, the playground component installs the latest stable version of these components, and then start these instances.
- Because this command does not specify the version of the TiDB, PD, and TiKV component, TiUP playground uses the latest version of each component by default. Assume that the latest version is v5.3.0, then this command works the same as `tiup ↳ playground:v1.7.0 v5.3.0`.
- Because this command does not specify the number of each component, TiUP playground, by default, starts a smallest cluster that consists of one TiDB instance, one TiKV instance, one PD instance, and one TiFlash instance.
- After starting each TiDB component, TiUP playground reminds you that the cluster is successfully started and provides you some useful information, such as how to connect to the TiDB cluster through the MySQL client and how to access the [TiDB Dashboard](#).

The command-line flags of the playground component are described as follows:

Flags:

<code>--db int</code>	Specify the number of TiDB instances (default: → 1)
<code>----db.host host</code>	Specify the listening address of TiDB
<code>----db.port int</code>	Specify the port of TiDB
<code>--db.binpath string</code>	Specify the TiDB instance binary path (optional → , for debugging)
<code>--db.config string</code>	Specify the TiDB instance configuration file ( → optional, for debugging)
<code>--db.timeout int</code>	Specify TiDB maximum <code>wait</code> time in seconds for → starting. 0 means no limit
<code>--drainer int</code>	Specify Drainer data of the cluster
<code>--drainer.binpath string</code>	Specify the location of the Drainer binary → files (optional, for debugging)
<code>--drainer.config string</code>	Specify the Drainer configuration file
<code>-h, --help</code>	<code>help</code> for tiup
<code>--host string</code>	Specify the listening address of each component → (default: `127.0.0.1`). Set it to `0.0.0.0` if provided for → access of other machines
<code>--kv int</code>	Specify the number of TiKV instances (default: → 1)

```
--kv.binpath string      Specify the TiKV instance binary path (optional
    ↪ , for debugging)
--kv.config string       Specify the TiKV instance configuration file (
    ↪ optional, for debugging)
--mode string            Specify the playground mode: 'tidb' (default)
    ↪ and 'tikv-slim'
--pd int                 Specify the number of PD instances (default: 1)
--pd.host host           Specify the listening address of PD
--pd.binpath string      Specify the PD instance binary path (optional,
    ↪ for debugging)
--pd.config string       Specify the PD instance configuration file (
    ↪ optional, for debugging)
--pump int               Specify the number of Pump instances. If the
    ↪ value is not `0`, TiDB Binlog is enabled.
--pump.binpath string    Specify the location of the Pump binary files (
    ↪ optional, for debugging)
--pump.config string     Specify the Pump configuration file (optional,
    ↪ for debugging)
-T, --tag string          Specify a tag for playground
--ticdc int               Specify the number of TiCDC instances (default:
    ↪ 0)
--ticdc.binpath string   Specify the TiCDC instance binary path (
    ↪ optional, for debugging)
--ticdc.config string     Specify the TiCDC instance configuration file (
    ↪ optional, for debugging)
--tiflash int              Specify the number of TiFlash instances (
    ↪ default: 1)
--tiflash.binpath string  Specify the TiFlash instance binary path (
    ↪ optional, for debugging)
--tiflash.config string    Specify the TiFlash instance configuration
    ↪ file (optional, for debugging)
--tiflash.timeout int     Specify TiFlash maximum wait time in seconds
    ↪ for starting. 0 means no limit
-v, --version             Specify the version of playground
--without-monitor         Disable the monitoring function of Prometheus
    ↪ and Grafana. If you do not add this flag, the monitoring
    ↪ function is enabled by default.
```

#### 10.4.10.1.2 Examples

Use the nightly version to start a TiDB cluster

```
tiup playground nightly
```

In the command above, `nightly` indicates the latest development version of TiDB.

Start a cluster with monitor

```
tiup playground nightly
```

This command starts Prometheus on port 9090 to display the time series data in the cluster.

Override PD's default configuration

First, you need to copy the [PD configuration template](#). Assume you place the copied file to `~/config/pd.toml` and make some changes according to your need, then you can execute the following command to override PD's default configuration:

```
tiup playground --pd.config ~/config/pd.toml
```

Replace the default binary files

By default, when playground is started, each component is started using the binary files from the official mirror. If you want to put a temporarily compiled local binary file into the cluster for testing, you can use the `--{comp}.binpath` flag for replacement. For example, execute the following command to replace the binary file of TiDB:

```
tiup playground --db.binpath /xx/tidb-server
```

Start multiple component instances

By default, only one instance is started for each TiDB, TiKV, and PD component. To start multiple instances for each component, add the following flag:

```
tiup playground v5.3.0 --db 3 --pd 3 --kv 3
```

#### 10.4.10.1.3 Quickly connect to the TiDB cluster started by playground

TiUP provides the `client` component, which is used to automatically find and connect to a local TiDB cluster started by playground. The usage is as follows:

```
tiup client
```

This command provides a list of TiDB clusters that are started by playground on the current machine on the console. Select the TiDB cluster to be connected. After clicking Enter, a built-in MySQL client is opened to connect to TiDB.

#### 10.4.10.1.4 View information of the started cluster

```
tiup playground display
```

The command above returns the following results:

Pid	Role	Uptime
84518	pd	35m22.929404512s
84519	tikv	35m22.927757153s
84520	pump	35m22.92618275s
86189	tidb	exited
86526	tidb	34m28.293148663s
86190	drainer	35m19.91349249s

#### 10.4.10.1.5 Scale out a cluster

The command-line parameter for scaling out a cluster is similar to that for starting a cluster. You can scale out two TiDB instances by executing the following command:

```
tiup playground scale-out --db 2
```

#### 10.4.10.1.6 Scale in a cluster

You can specify a pid in the `tiup playground scale-in` command to scale in the corresponding instance. To view the pid, execute `tiup playground display`.

```
tiup playground scale-in --pid 86526
```

### 10.4.10.2 Deploy and Maintain an Online TiDB Cluster Using TiUP

This document focuses on how to use the TiUP cluster component. For the complete steps of online deployment, refer to [Deploy a TiDB Cluster Using TiUP](#).

Similar to [the TiUP playground component](#) used for a local test deployment, the TiUP cluster component quickly deploys TiDB for production environment. Compared with playground, the cluster component provides more powerful production cluster management features, including upgrading, scaling, and even operation and auditing.

For the help information of the cluster component, run the following command:

```
tiup cluster
```

```
Starting component `cluster`: /home/tidb/.tiup/components/cluster/v1.3.0/
  ↳ cluster
```

```
Deploy a TiDB cluster for production
```

Usage:

```
cluster [flags]
cluster [command]
```

Available Commands:

check	Perform preflight checks for the cluster
deploy	Deploy a cluster for production
start	Start a TiDB cluster
stop	Stop a TiDB cluster
restart	Restart a TiDB cluster
scale-in	Scale in a TiDB cluster
scale-out	Scale out a TiDB cluster
clean	Clean up cluster data
destroy	Destroy a specified cluster
upgrade	Upgrade a specified TiDB cluster
exec	Run shell command on host in the tidb cluster
display	Display information of a TiDB cluster
list	List all clusters
audit	Show audit log of cluster operation
import	Import an exist TiDB cluster from TiDB Ansible
edit-config	Edit TiDB cluster config
reload	Reload a TiDB cluster's config and restart if needed
patch	Replace the remote package with a specified package and restart → the service
help	Help about any command

Flags:

-h, --help	help for cluster
--native-ssh	Use the system's native SSH client
--wait-timeout int	Timeout of waiting the operation
--ssh-timeout int	Timeout in seconds to connect host via SSH, ignored → for operations that don't need an SSH connection. (default 5)
-y, --yes	Skip all confirmations and assumes 'yes'

#### 10.4.10.2.1 Deploy the cluster

To deploy the cluster, run the `tiup cluster deploy` command. The usage of the command is as follows:

```
tiup cluster deploy <cluster-name> <version> <topology.yaml> [flags]
```

This command requires you to provide the cluster name, the TiDB cluster version, and a topology file of the cluster.

To write a topology file, refer to [the example](#). The following file is an example of the simplest topology:

**Note:**

The topology file used by the TiUP cluster component for deployment and scaling is written using [yaml](#) syntax, so make sure that the indentation is correct.

```
---
pd_servers:
  - host: 172.16.5.134
    name: pd-134
  - host: 172.16.5.139
    name: pd-139
  - host: 172.16.5.140
    name: pd-140

tidb_servers:
  - host: 172.16.5.134
  - host: 172.16.5.139
  - host: 172.16.5.140

tikv_servers:
  - host: 172.16.5.134
  - host: 172.16.5.139
  - host: 172.16.5.140

grafana_servers:
  - host: 172.16.5.134

monitoring_servers:
  - host: 172.16.5.134
```

By default, TiUP is deployed as the binary files running on the amd64 architecture. If the target machine is the arm64 architecture, you can configure it in the topology file:

```
global:
  arch: "arm64"          # Configures all machines to use the binary files of
                         # the arm64 architecture by default

tidb_servers:
  - host: 172.16.5.134
    arch: "amd64"          # Configures this machine to use the binary files of
                           # the amd64 architecture
  - host: 172.16.5.139
```

```

arch: "arm64"      # Configures this machine to use the binary files of
                   ↪ the arm64 architecture
- host: 172.16.5.140 # Machines that are not configured with the arch
                   ↪ field use the default value in the global field, which is arm64 in
                   ↪ this case.

...

```

Save the file as `/tmp/topology.yaml`. If you want to use TiDB v3.0.12 and your cluster name is `prod-cluster`, run the following command:

```
tiup cluster deploy -p prod-cluster v3.0.12 /tmp/topology.yaml
```

During the execution, TiUP asks you to confirm your topology again and requires the root password of the target machine (the `-p` flag means inputting password):

```

Please confirm your topology:
TiDB Cluster: prod-cluster
TiDB Version: v3.0.12
Type      Host        Ports      Directories
----      ----        -----      -----
pd        172.16.5.134 2379/2380  deploy/pd-2379,data/pd-2379
pd        172.16.5.139 2379/2380  deploy/pd-2379,data/pd-2379
pd        172.16.5.140 2379/2380  deploy/pd-2379,data/pd-2379
tikv     172.16.5.134 20160/20180 deploy/tikv-20160,data/tikv-20160
tikv     172.16.5.139 20160/20180 deploy/tikv-20160,data/tikv-20160
tikv     172.16.5.140 20160/20180 deploy/tikv-20160,data/tikv-20160
tidb     172.16.5.134 4000/10080  deploy/tidb-4000
tidb     172.16.5.139 4000/10080  deploy/tidb-4000
tidb     172.16.5.140 4000/10080  deploy/tidb-4000
prometheus 172.16.5.134 9090      deploy/prometheus-9090,data/prometheus
                   ↪ -9090
grafana   172.16.5.134 3000      deploy/grafana-3000
Attention:
 1. If the topology is not what you expected, check your yaml file.
 2. Please confirm there is no port/directory conflicts in same host.
Do you want to continue? [y/N] :

```

After you enter the password, TiUP cluster downloads the required components and deploy them on the corresponding machines. When you see the following message, the deployment is successful:

```
Deployed cluster `prod-cluster` successfully
```

#### 10.4.10.2.2 View the cluster list

After the cluster is successfully deployed, view the cluster list by running the following command:

```
tiup cluster list
```

```
Starting /root/.tiup/components/cluster/v1.3.0/cluster list
Name      User Version  Path
  ↵ PrivateKey
-----
  ↵ -----
prod-cluster tidb v3.0.12  /root/.tiup/storage/cluster/clusters/prod-
  ↵ cluster /root/.tiup/storage/cluster/clusters/prod-cluster/ssh/id_rsa
```

#### 10.4.10.2.3 Start the cluster

After the cluster is successfully deployed, start the cluster by running the following command:

```
tiup cluster start prod-cluster
```

If you forget the name of your cluster, view the cluster list by running `tiup cluster list`.

#### 10.4.10.2.4 Check the cluster status

TiUP provides the `tiup cluster display` command to view the status of each component in the cluster. With this command, you don't have to log in to each machine to see the component status. The usage of the command is as follows:

```
tiup cluster display prod-cluster
```

```
Starting /root/.tiup/components/cluster/v1.3.0/cluster display prod-cluster
TiDB Cluster: prod-cluster
TiDB Version: v3.0.12
ID          Role      Host       Ports     Status   Data Dir
  ↵          Deploy Dir
--          ----      ----       -----    -----   -----
  ↵          -----
172.16.5.134:3000 grafana   172.16.5.134 3000      Up      -
  ↵          deploy/grafana-3000
172.16.5.134:2379 pd        172.16.5.134 2379/2380 Up|L     data/pd-2379
  ↵          deploy/pd-2379
172.16.5.139:2379 pd        172.16.5.139 2379/2380 Up|UI    data/pd-2379
  ↵          deploy/pd-2379
172.16.5.140:2379 pd        172.16.5.140 2379/2380 Up      data/pd-2379
  ↵          deploy/pd-2379
```

172.16.5.134:9090	prometheus	172.16.5.134	9090	Up	data/
↳ prometheus-9090	deploy/prometheus-9090				
172.16.5.134:4000	tidb	172.16.5.134	4000/10080	Up	-
↳	deploy/tidb-4000				
172.16.5.139:4000	tidb	172.16.5.139	4000/10080	Up	-
↳	deploy/tidb-4000				
172.16.5.140:4000	tidb	172.16.5.140	4000/10080	Up	-
↳	deploy/tidb-4000				
172.16.5.134:20160	tikv	172.16.5.134	20160/20180	Up	data/tikv
↳ -20160	deploy/tikv-20160				
172.16.5.139:20160	tikv	172.16.5.139	20160/20180	Up	data/tikv
↳ -20160	deploy/tikv-20160				
172.16.5.140:20160	tikv	172.16.5.140	20160/20180	Up	data/tikv
↳ -20160	deploy/tikv-20160				

The Status column uses Up or Down to indicate whether the service is running normally.

For the PD component, |L or |UI might be appended to Up or Down. |L indicates that the PD node is a Leader, and |UI indicates that [TiDB Dashboard](#) is running on the PD node.

#### 10.4.10.2.5 Scale in a cluster

##### Note:

This section describes only the syntax of the scale-in command. For detailed steps of online scaling, refer to [Scale the TiDB Cluster Using TiUP](#).

Scaling in a cluster means making some node(s) offline. This operation removes the specific node(s) from the cluster and deletes the remaining files.

Because the offline process of the TiKV and TiDB Binlog components is asynchronous (which requires removing the node through API), and the process takes a long time (which requires continuous observation on whether the node is successfully taken offline), special treatment is given to the TiKV and TiDB Binlog components.

- For TiKV and Binlog:
  - TiUP cluster takes the node offline through API and directly exits without waiting for the process to be completed.
  - Afterwards, when a command related to the cluster operation is executed, TiUP cluster examines whether there is a TiKV/Binlog node that has been taken offline. If not, TiUP cluster continues with the specified operation; If there is, TiUP cluster takes the following steps:

1. Stop the service of the node that has been taken offline.
  2. Clean up the data files related to the node.
  3. Remove the node from the cluster topology.
- For other components:
    - When taking the PD component down, TiUP cluster quickly deletes the specified node from the cluster through API, stops the service of the specified PD node, and deletes the related data files.
    - When taking other components down, TiUP cluster directly stops the node service and deletes the related data files.

The basic usage of the scale-in command:

```
tiup cluster scale-in <cluster-name> -N <node-id>
```

To use this command, you need to specify at least two flags: the cluster name and the node ID. The node ID can be obtained by using the `tiup cluster display` command in the previous section.

For example, to make the TiKV node on 172.16.5.140 offline, run the following command:

```
tiup cluster scale-in prod-cluster -N 172.16.5.140:20160
```

By running `tiup cluster display`, you can see that the TiKV node is marked Offline:

```
tiup cluster display prod-cluster
```

```
Starting /root/.tiup/components/cluster/v1.3.0/cluster display prod-cluster
TiDB Cluster: prod-cluster
TiDB Version: v3.0.12
ID           Role     Host       Ports      Status    Data Dir
--           Deploy Dir
---          ----   ---       -----      -----
172.16.5.134:3000 grafana  172.16.5.134 3000      Up        -
172.16.5.134:2379 pd      172.16.5.134 2379/2380 Up|L      data/pd-2379
172.16.5.139:2379 pd      172.16.5.139 2379/2380 Up|UI     data/pd-2379
172.16.5.140:2379 pd      172.16.5.140 2379/2380 Up        data/pd-2379
172.16.5.134:9090 prometheus 172.16.5.134 9090      Up        data/
172.16.5.134:9090 prometheus 172.16.5.134 9090      Up        data/
```

172.16.5.134:4000 tidb	172.16.5.134	4000/10080	Up	-
↪	deploy/tidb-4000			
172.16.5.139:4000 tidb	172.16.5.139	4000/10080	Up	-
↪	deploy/tidb-4000			
172.16.5.140:4000 tidb	172.16.5.140	4000/10080	Up	-
↪	deploy/tidb-4000			
172.16.5.134:20160 tikv	172.16.5.134	20160/20180	Up	data/tikv
↪ -20160	deploy/tikv-20160			
172.16.5.139:20160 tikv	172.16.5.139	20160/20180	Up	data/tikv
↪ -20160	deploy/tikv-20160			
172.16.5.140:20160 tikv	172.16.5.140	20160/20180	Offline	data/tikv
↪ -20160	deploy/tikv-20160			

After PD schedules the data on the node to other TiKV nodes, this node will be deleted automatically.

#### 10.4.10.2.6 Scale out a cluster

**Note:**

This section describes only the syntax of the scale-out command. For detailed steps of online scaling, refer to [Scale the TiDB Cluster Using TiUP](#).

The scale-out operation has an inner logic similar to that of deployment: the TiUP cluster component firstly ensures the SSH connection of the node, creates the required directories on the target node, then executes the deployment operation, and starts the node service.

When you scale out PD, the node is added to the cluster by `join`, and the configurations of services associated with PD are updated. When you scale out other services, the service is started directly and added to the cluster.

All services conduct correctness validation when they are scaled out. The validation results show whether the scaling-out is successful.

To add a TiKV node and a PD node in the `tidb-test` cluster, take the following steps:

1. Create a `scale.yaml` file, and add IPs of the new TiKV and PD nodes:

**Note:**

You need to create a topology file, which includes only the description of the new nodes, not the existing nodes.

```
---
pd_servers:
  - ip: 172.16.5.140

tikv_servers:
  - ip: 172.16.5.140
```

2. Perform the scale-out operation. TiUP cluster adds the corresponding nodes to the cluster according to the port, directory, and other information described in `scale.yaml`.

```
tiup cluster scale-out tidb-test scale.yaml
```

After the command is executed, you can check the status of the scaled-out cluster by running `tiup cluster display tidb-test`.

#### 10.4.10.2.7 Rolling upgrade

##### Note:

This section describes only the syntax of the upgrade command. For detailed steps of online upgrade, refer to [Upgrade TiDB Using TiUP](#).

The rolling upgrade feature leverages the distributed capabilities of TiDB. The upgrade process is made as transparent as possible to the application, and does not affect the business.

Before the upgrade, TiUP cluster checks whether the configuration file of each component is rational. If so, the components are upgraded node by node; if not, TiUP reports an error and exits. The operations vary with different nodes.

##### Operations for different nodes

- Upgrade the PD node
  - First, upgrade non-Leader nodes.
  - After all the non-Leader nodes are upgraded, upgrade the Leader node.
    - \* The upgrade tool sends a command to PD that migrates Leader to an already upgraded node.
    - \* After the Leader role is switched to another node, upgrade the previous Leader node.

- During the upgrade, if any unhealthy node is detected, the tool stops this upgrade operation and exits. You need to manually analyze the cause, fix the issue and run the upgrade again.
- Upgrade the TiKV node
  - First, add a scheduling operation in PD that migrates the Region Leader of this TiKV node. This ensures that the upgrade process does not affect the business.
  - After the Leader is migrated, upgrade this TiKV node.
  - After the upgraded TiKV is started normally, remove the scheduling of the Leader.
- Upgrade other services
  - Stop the service normally and update the node.

Upgrade command

The flags for the upgrade command is as follows:

Usage:

```
cluster upgrade <cluster-name> <version> [flags]
```

Flags:

--force	Force upgrade won't transfer leader
-h, --help	help for upgrade
--transfer-timeout int	Timeout in seconds when transferring PD and → TiKV store leaders (default 300)

Global Flags:

--native-ssh	Use the system's native SSH client
--wait-timeout int	Timeout of waiting the operation
--ssh-timeout int	Timeout in seconds to connect host via SSH, ignored → for operations that don't need an SSH connection. (default 5)
-y, --yes	Skip all confirmations and assumes 'yes'

For example, the following command upgrades the cluster to v5.0.0:

```
tiup cluster upgrade tidb-test v5.0.0
```

#### 10.4.10.2.8 Update configuration

If you want to dynamically update the component configurations, the TiUP cluster component saves a current configuration for each cluster. To edit this configuration, execute the `tiup cluster edit-config <cluster-name>` command. For example:

```
tiup cluster edit-config prod-cluster
```

TiUP cluster opens the configuration file in the vi editor. If you want to use other editors, use the `EDITOR` environment variable to customize the editor, such as `export EDITOR=nano`.

After editing the file, save the changes. To apply the new configuration to the cluster, execute the following command:

```
tiup cluster reload prod-cluster
```

The command sends the configuration to the target machine and restarts the cluster to make the configuration take effect.

#### Note:

For monitoring components, customize the configuration by executing the `tiup cluster edit-config` command to add a custom configuration path on the corresponding instance. For example:

```
---
```

```
grafana_servers:
  - host: 172.16.5.134
    dashboard_dir: /path/to/local/dashboards/dir

monitoring_servers:
  - host: 172.16.5.134
    rule_dir: /path/to/local/rules/dir

alertmanager_servers:
  - host: 172.16.5.134
    config_file: /path/to/local/alertmanager.yml
```

The content and format requirements for files under the specified path are as follows:

- The folder specified in the `dashboard_dir` field of `grafana_servers` must contain full `*.json` files.
- The folder specified in the `rule_dir` field of `monitoring_servers` must contain full `*.rules.yml` files.
- For the format of files specified in the `config_file` field of `alertmanager_servers`, refer to [the Alertmanager configuration template](#).

When you execute `tiup reload`, TiUP first deletes all old configuration files in the target machine and then uploads the corresponding configuration from the control machine

to the corresponding configuration directory of the target machine. Therefore, if you want to modify a particular configuration file, make sure that all configuration files (including the unmodified ones) are in the same directory. For example, to modify Grafana's `tidb.json` file, you need to first copy all the `*.json` files from Grafana's `dashboards` directory to your local directory. Otherwise, other JSON files will be missing from the target machine.

#### Note:

If you have configured the `dashboard_dir` field of `grafana_servers`, after executing the `tiup cluster rename` command to rename the cluster, you need to complete the following operations:

1. In the local `dashboards` directory, change the cluster name to the new cluster name.
2. In the local `dashboards` directory, change `datasource` to the new cluster name, because `datasource` is named after the cluster name.
3. Execute the `tiup cluster reload -R grafana` command.

#### 10.4.10.2.9 Update component

For normal upgrade, you can use the `upgrade` command. But in some scenarios, such as debugging, you might need to replace the currently running component with a temporary package. To achieve this, use the `patch` command:

```
tiup cluster patch --help
```

Replace the remote package with a specified package and restart the service

##### Usage:

```
cluster patch <cluster-name> <package-path> [flags]
```

##### Flags:

<code>-h, --help</code>	help for patch
<code>-N, --node strings</code>	Specify the nodes
<code>--overwrite</code>	Use this package in the future scale-out → operations
<code>-R, --role strings</code>	Specify the role --transfer-timeout int Timeout in seconds when transferring PD and → TiKV store leaders (default 300)

##### Global Flags:

<code>--native-ssh</code>	Use the system's native SSH client
<code>--wait-timeout int</code>	Timeout of waiting the operation

```
--ssh-timeout int Timeout in seconds to connect host via SSH, ignored
    ↪ for operations that don't need an SSH connection. (default 5)
-y, --yes           Skip all confirmations and assumes 'yes'
```

If a TiDB hotfix package is in `/tmp/tidb-hotfix.tar.gz` and you want to replace all the TiDB packages in the cluster, run the following command:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -R tidb
```

You can also replace only one TiDB package in the cluster:

```
tiup cluster patch test-cluster /tmp/tidb-hotfix.tar.gz -N 172.16.4.5:4000
```

#### 10.4.10.2.10 Import TiDB Ansible cluster

##### Note:

Currently, TiUP cluster's support for TiSpark is still **experimental**. It is not supported to import a TiDB cluster with TiSpark enabled.

Before TiUP is released, TiDB Ansible is often used to deploy TiDB clusters. To enable TiUP to take over the cluster deployed by TiDB Ansible, use the `import` command.

The usage of the `import` command is as follows:

```
tiup cluster import --help
```

Import an exist TiDB cluster from TiDB-Ansible

##### Usage:

```
cluster import [flags]
```

##### Flags:

```
-d, --dir string      The path to TiDB-Ansible directory
-h, --help            help for import
--inventory string   The name of inventory file (default "inventory.ini"
    ↪ ")
--no-backup          Don't backup ansible dir, useful when there're
    ↪ multiple inventory files
-r, --rename NAME    Rename the imported cluster to NAME
```

##### Global Flags:

```
--native-ssh         Use the system's native SSH client
```

```
--wait-timeout int Timeout of waiting the operation
--ssh-timeout int Timeout in seconds to connect host via SSH, ignored
    ↪ for operations that don't need an SSH connection. (default 5)
-y, --yes           Skip all confirmations and assumes 'yes'
```

You can use either of the following commands to import a TiDB Ansible cluster:

```
cd tidb-ansible
tiup cluster import
```

```
tiup cluster import --dir=/path/to/tidb-ansible
```

#### 10.4.10.2.11 View the operation log

To view the operation log, use the `audit` command. The usage of the `audit` command is as follows:

Usage:

```
tiup cluster audit [audit-id] [flags]
```

Flags:

```
-h, --help help for audit
```

If the `[audit-id]` flag is not specified, the command shows a list of commands that have been executed. For example:

```
tiup cluster audit
```

```
Starting component `cluster`: /home/tidb/.tiup/components/cluster/v1.3.0/
    ↪ cluster audit
ID      Time                Command
--      ----
4BLhr0 2020-04-29T13:25:09+08:00 /home/tidb/.tiup/components/cluster/v1.3.0/
    ↪ cluster deploy test v5.0.0-rc /tmp/topology.yaml
4BKWjF 2020-04-28T23:36:57+08:00 /home/tidb/.tiup/components/cluster/v1.3.0/
    ↪ cluster deploy test v5.0.0-rc /tmp/topology.yaml
4BKWhH 2020-04-28T23:02:08+08:00 /home/tidb/.tiup/components/cluster/v1.3.0/
    ↪ cluster deploy test v5.0.0-rc /tmp/topology.yaml
4BKKH1 2020-04-28T16:39:04+08:00 /home/tidb/.tiup/components/cluster/v1.3.0/
    ↪ cluster destroy test
4BKKDx 2020-04-28T16:36:57+08:00 /home/tidb/.tiup/components/cluster/v1.3.0/
    ↪ cluster deploy test v5.0.0-rc /tmp/topology.yaml
```

The first column is `audit-id`. To view the execution log of a certain command, pass the `audit-id` of a command as the flag as follows:

```
tiup cluster audit 4BLhr0
```

#### 10.4.10.2.12 Run commands on a host in the TiDB cluster

To run command on a host in the TiDB cluster, use the `exec` command. The usage of the `exec` command is as follows:

Usage:

```
cluster exec <cluster-name> [flags]
```

Flags:

```
--command string the command run on cluster host (default "ls")
-h, --help      help for exec
-N, --node strings Only exec on host with specified nodes
-R, --role strings Only exec on host with specified roles
--sudo          use root permissions (default false)
```

Global Flags:

```
--ssh-timeout int Timeout in seconds to connect host via SSH, ignored
    ↪ for operations that don't need an SSH connection. (default 5)
-y, --yes        Skip all confirmations and assumes 'yes'
```

For example, to execute `ls /tmp` on all TiDB nodes, run the following command:

```
tiup cluster exec test-cluster --command='ls /tmp'
```

#### 10.4.10.2.13 Cluster controllers

Before TiUP is released, you can control the cluster using `tidb-ctl`, `tikv-ctl`, `pd-ctl`, and other tools. To make the tools easier to download and use, TiUP integrates them into an all-in-one component, `ctl`.

Usage:

```
tiup ctl {tidb/pd/tikv/binlog/etc} [flags]
```

Flags:

```
-h, --help help for tiup
```

This command has a corresponding relationship with those of the previous tools:

<code>tidb-ctl [args]</code>	<code>= tiup ctl tidb [args]</code>
<code>pd-ctl [args]</code>	<code>= tiup ctl pd [args]</code>
<code>tikv-ctl [args]</code>	<code>= tiup ctl tikv [args]</code>
<code>binlogctl [args]</code>	<code>= tiup ctl bindlog [args]</code>
<code>etcctl [args]</code>	<code>= tiup ctl etcd [args]</code>

For example, if you previously view the store by running `pd-ctl -u http://127.0.0.1:2379 ↪ store`, now you can run the following command in TiUP:

```
tiup ctl pd -u http://127.0.0.1:2379 store
```

#### 10.4.10.2.14 Environment checks for target machines

You can use the `check` command to perform a series of checks on the environment of the target machine and output the check results. By executing the `check` command, you can find common unreasonable configurations or unsupported situations. The command flag list is as follows:

<b>Usage:</b>	<code>tiup cluster check &lt;topology.yml   cluster-name&gt; [flags]</code>
<b>Flags:</b>	
<code>--apply</code>	Try to fix failed checks
<code>--cluster</code>	Check existing cluster, the input is a cluster → name.
<code>--enable-cpu</code>	Enable CPU thread count check
<code>--enable-disk</code>	Enable disk IO (fio) check
<code>--enable-mem</code>	Enable memory size check
<code>-h, --help</code>	<b>help for check</b>
<code>-i, --identity_file</code> string	The path of the SSH identity file. If specified → , public key authentication will be used.
<code>-p, --password</code>	Use password of target hosts. If specified, → password authentication will be used.
<code>--user</code> string	The user name to <b>login</b> via SSH. The user must → has root (or sudo) privilege.

By default, this command is used to check the environment before deployment. By specifying the `--cluster` flag to switch the mode, you can also check the target machines of an existing cluster, for example:

```
#### check deployed servers before deployment
tiup cluster check topology.yml --user tidb -p
#### check deployed servers of an existing cluster
tiup cluster check <cluster-name> --cluster
```

The CPU thread count check, memory size check, and disk performance check are disabled by default. For the production environment, it is recommended that you enable the three checks and make sure they pass to obtain the best performance.

- CPU: If the number of threads is greater than or equal to 16, the check is passed.
- Memory: If the total size of physical memory is greater than or equal to 32 GB, the check is passed.
- Disk: Execute `fio` test on the partitions of `data_dir` and record the results.

When running the checks, if the `--apply` flag is specified, the program automatically repairs the failed items. Automatic repair is limited to some items that can be adjusted by modifying the configuration or system parameters. Other unrepaired items need to be handled manually according to the actual situation.

Environment checks are not necessary for deploying a cluster. For the production environment, it is recommended to perform environment checks and pass all check items before deployment. If not all the check items are passed, the cluster might be deployed and run normally, but the best performance might not be obtained.

#### 10.4.10.2.15 Use the system's native SSH client to connect to cluster

All operations above performed on the cluster machine use the SSH client embedded in TiUP to connect to the cluster and execute commands. However, in some scenarios, you might also need to use the SSH client native to the control machine system to perform such cluster operations. For example:

- To use a SSH plug-in for authentication
- To use a customized SSH client

Then you can use the `--native-ssh` command-line flag to enable the system-native command-line tool:

- Deploy a cluster: `tiup cluster deploy <cluster-name> <version> <topo> --native-ssh`
- Start a cluster: `tiup cluster start <cluster-name> --native-ssh`
- Upgrade a cluster: `tiup cluster upgrade ... --native-ssh`

You can add `--native-ssh` in all cluster operation commands above to use the system's native SSH client.

To avoid adding such a flag in every command, you can use the `TIUP_NATIVE_SSH` system variable to specify whether to use the local SSH client:

```
export TIUP_NATIVE_SSH=true
#### or
export TIUP_NATIVE_SSH=1
#### or
export TIUP_NATIVE_SSH=enable
```

If you specify this environment variable and `--native-ssh` at the same time, `--native-ssh` has higher priority.

**Note:**

During the process of cluster deployment, if you need to use a password for connection (`-p`) or passphrase is configured in the key file, you must ensure that `sshpass` is installed on the control machine; otherwise, a timeout error is reported.

#### 10.4.10.2.16 Migrate control machine and back up TiUP data

The TiUP data is stored in the `.tiup` directory in the user's home directory. To migrate the control machine, you can take the following steps to copy the `.tiup` directory to the corresponding target machine:

1. Execute `tar czvf tiup.tar.gz .tiup` in the home directory of the original machine.
2. Copy `tiup.tar.gz` to the home directory of the target machine.
3. Execute `tar xzvf tiup.tar.gz` in the home directory of the target machine.
4. Add the `.tiup` directory to the `PATH` environment variable.

If you use `bash` and you are a `tidb` user, you can add `export PATH=/home/tidb/.tiup/bin:$PATH` in `~/.bashrc` and execute `source ~/.bashrc`. Then make corresponding adjustments according to the shell and the user you use.

##### Note:

It is recommended that you back up the `.tiup` directory regularly to avoid the loss of TiUP data caused by abnormal conditions, such as disk damage of the control machine.

#### 10.4.10.3 Create a Private Mirror

When creating a private cloud, usually, you need to use an isolated network environment, where the official mirror of TiUP is not accessible. Therefore, you can create a private mirror, which is mainly implemented by the `mirror` command. You can also use the `mirror` command for offline deployment.

##### 10.4.10.3.1 TiUP mirror overview

Execute the following command to get the help information of the `mirror` command:

```
tiup mirror --help
```

The '`mirror`' `command` is used to manage a component repository `for` TiUP, you  
can use it to create a private repository, or to add new component to an existing  
repository.  
The repository can be used either online or offline.  
It also provides some useful utilities to `help` managing keys, users and  
versions of components or the repository itself.

```

Usage:
  tiup mirror <command> [flags]
Available Commands:
  init      Initialize an empty repository
  sign      Add signatures to a manifest file
  genkey    Generate a new key pair
  clone     Clone a local mirror from remote mirror and download all
            → selected components
  publish   Publish a component
Flags:
  -h, --help      help for mirror
      --repo string Path to the repository
Global Flags:
  --skip-version-check Skip the strict version check, by default a
            → version must be a valid SemVer string
Use "tiup mirror [command] --help" for more information about a command.

```

The `tiup mirror clone` command is used to build a local mirror. The basic usage is as follows:

```
tiup mirror clone <target-dir> [global-version] [flags]
```

- `target-dir`: used to specify the directory in which cloned data is stored.
- `global-version`: used to quickly set a global version for all components.

The `tiup mirror clone` command provides many optional flags (might provide more in the future). These flags can be divided into the following categories according to their intended usages:

- Determines whether to use prefix matching to match the version when cloning  
If the `--prefix` flag is specified, the version number is matched by prefix for the clone. For example, if you specify `--prefix` as “v5.0.0”, then “v5.0.0-rc”, and “v5.0.0” are matched.
- Determines whether to use the full clone  
If you specify the `--full` flag, you can clone the official mirror fully.

#### Note:

If `--full`, `global-version` flags, and the component versions are not specified, only some meta information is cloned.

- Determines whether to clone packages from the specific platform

If you want to clone packages only for a specific platform, use `--os` and `--arch` to specify the platform. For example:

- Execute the `tiup mirror clone <target-dir> [global-version] --os=linux` command to clone for linux.
- Execute the `tiup mirror clone <target-dir> [global-version] --arch=amd64` command to clone for amd64.
- Execute the `tiup mirror clone <target-dir> [global-version] --os=linux --arch=amd64` command to clone for linux/amd64.

- Determines whether to clone a specific version of a package

If you want to clone only one version (not all versions) of a component, use `--<component>=<version>` to specify this version. For example:

- Execute the `tiup mirror clone <target-dir> --tidb v5.3.0` command to clone the v5.3.0 version of the TiDB component.
- Execute the `tiup mirror clone <target-dir> --tidb v5.3.0 --tikv all` command to clone the v5.3.0 version of the TiDB component and all versions of the TiKV component.
- Execute the `tiup mirror clone <target-dir> v5.3.0` command to clone the v5.3.0 version of all components in a cluster.

#### 10.4.10.3.2 Usage examples

This section introduces the usage examples of the `mirror` command.

The repository that is cloned using `tiup mirror clone` can be shared among hosts either by sharing the files via SCP, NFS or by making the repository available over the HTTP or HTTPS protocol. Use `tiup mirror set <location>` to specify the location of the repository.

Refer to [Deploy a TiDB Cluster Using TiUP](#) to install the TiUP offline mirror, deploy a TiDB cluster, and start it.

#### 10.4.10.4 Stress Test TiDB Using TiUP Bench Component

When you test the performance of a database, it is often required to stress test the database. To facilitate this, TiUP has integrated the bench component, which provides two workloads for stress testing: TPC-C and TPC-H. The commands and flags are as follows:

```
tiup bench
```

```
Starting component `bench`: /home/tidb/.tiup/components/bench/v1.5.0/bench
Benchmark database with different workloads
```

Usage:

```

tiup bench [command]

Available Commands:
  help      Help about any command
  tpcc
  tpch

Flags:
  --count int          Total execution count, 0 means infinite
  -D, --db string      Database name (default "test")
  -d, --driver string   Database driver: mysql
  --dropdata            Cleanup data before prepare
  -h, --help             help for /Users/joshua/.tiup/components/bench/v0
                        ↪ .0.1/bench
  -H, --host string     Database host (default "127.0.0.1")
  --ignore-error        Ignore error when running workload
  --interval duration   Output interval time (default 10s)
  --isolation int       Isolation Level 0: Default, 1: ReadUncommitted,
                        2: ReadCommitted, 3: WriteCommitted, 4:
                        ↪ RepeatableRead,
                        5: Snapshot, 6: Serializable, 7: Linerizable
  --max-procs int       runtime.GOMAXPROCS
  -p, --password string Database password
  -P, --port int         Database port (default 4000)
  --pprof string         Address of pprof endpoint
  --silence              Don't print error when running workload
  --summary               Print summary TPM only, or also print current TPM
                        ↪ when running workload
  -T, --threads int      Thread concurrency (default 16)
  --time duration         Total execution time (default 2562047h47m16
                        ↪ .854775807s)
  -U, --user string       Database user (default "root")

```

The following sections describe how to run TPC-C and TPC-H tests using TiUP.

#### 10.4.10.4.1 Run TPC-C test using TiUP

The TiUP bench component supports the following commands and flags to run the TPC-C test:

```

Available Commands:
  check      Check data consistency for the workload
  cleanup    Cleanup data for the workload
  prepare    Prepare data for the workload
  run        Run workload

```

Flags:

```
--check-all      Run all consistency checks
-h, --help       help for tpcc
--output string  Output directory for generating csv file when
                 → preparing data
--parts int      Number to partition warehouses (default 1)
--tables string  Specified tables for generating file, separated by
                 → ','. Valid only if output is set. If this flag is not set,
                 → generate all tables by default.
--warehouses int Number of warehouses (default 10)
```

Test procedures

1. Create 4 warehouses using 4 partitions via hash:

```
tiup bench tpcc --warehouses 4 --parts 4 prepare
```

2. Run the TPC-C test:

```
tiup bench tpcc --warehouses 4 run
```

3. Clean up data:

```
tiup bench tpcc --warehouses 4 cleanup
```

4. Check the consistency:

```
tiup bench tpcc --warehouses 4 check
```

5. Generate the CSV file:

```
tiup bench tpcc --warehouses 4 prepare --output data
```

6. Generate the CSV file for the specified table:

```
tiup bench tpcc --warehouses 4 prepare --output data --tables history,
                 → orders
```

7. Enable pprof:

```
tiup bench tpcc --warehouses 4 prepare --output data --pprof :10111
```

#### 10.4.10.4.2 Run TPC-H test using TiUP

The TiUP bench component supports the following commands and parameters to run the TPC-H test:

Available Commands:

```
cleanup   Cleanup data for the workload
prepare   Prepare data for the workload
run       Run workload
```

Flags:

```
--check           Check output data, only when the scale factor equals 1
-h, --help        help for tpch
--queries string All queries (default "q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,
                  ↪ q11,q12,q13,q14,q15,q16,q17,q18,q19,q20,q21,q22")
--sf int         scale factor
```

Test procedures

1. Prepare data:

```
tiup bench tpch --sf=1 prepare
```

2. Run the TPC-H test by executing one of the following commands:

- If you check the result, run this command:

```
tiup bench tpch --sf=1 --check=true run
```

- If you do not check the result, run this command:

```
tiup bench tpch --sf=1 run
```

3. Clean up data:

```
tiup bench tpch cleanup
```

## 10.5 TiDB Operator

TiDB Operator is an automatic operation system for TiDB clusters in Kubernetes. It provides full life-cycle management for TiDB including deployment, upgrades, scaling, backup, fail-over, and configuration changes. With TiDB Operator, TiDB can run seamlessly in the Kubernetes clusters deployed on a public or private cloud.

For the current TiDB release, the compatible version of TiDB Operator is v1.2.

Currently, the TiDB Operator documentation (also named as TiDB in Kubernetes documentation) is independent of the TiDB documentation. To access the documentation, click the following link:

- TiDB in Kubernetes documentation.

## 10.6 Backup & Restore (BR)

### 10.6.1 BR Tool Overview

BR (Backup & Restore) is a command-line tool for distributed backup and restoration of the TiDB cluster data.

Compared with [dumpling](#), BR is more suitable for scenarios involved huge data volumes.

In addition to regular backup and restoration, you can also use BR for large-scale data migration as long as compatibility is ensured.

This document describes BR's implementation principles, recommended deployment configuration, usage restrictions and several methods to use BR.

#### 10.6.1.1 Implementation principles

BR sends the backup or restoration commands to each TiKV node. After receiving these commands, TiKV performs the corresponding backup or restoration operations.

Each TiKV node has a path in which the backup files generated in the backup operation are stored and from which the stored backup files are read during the restoration.

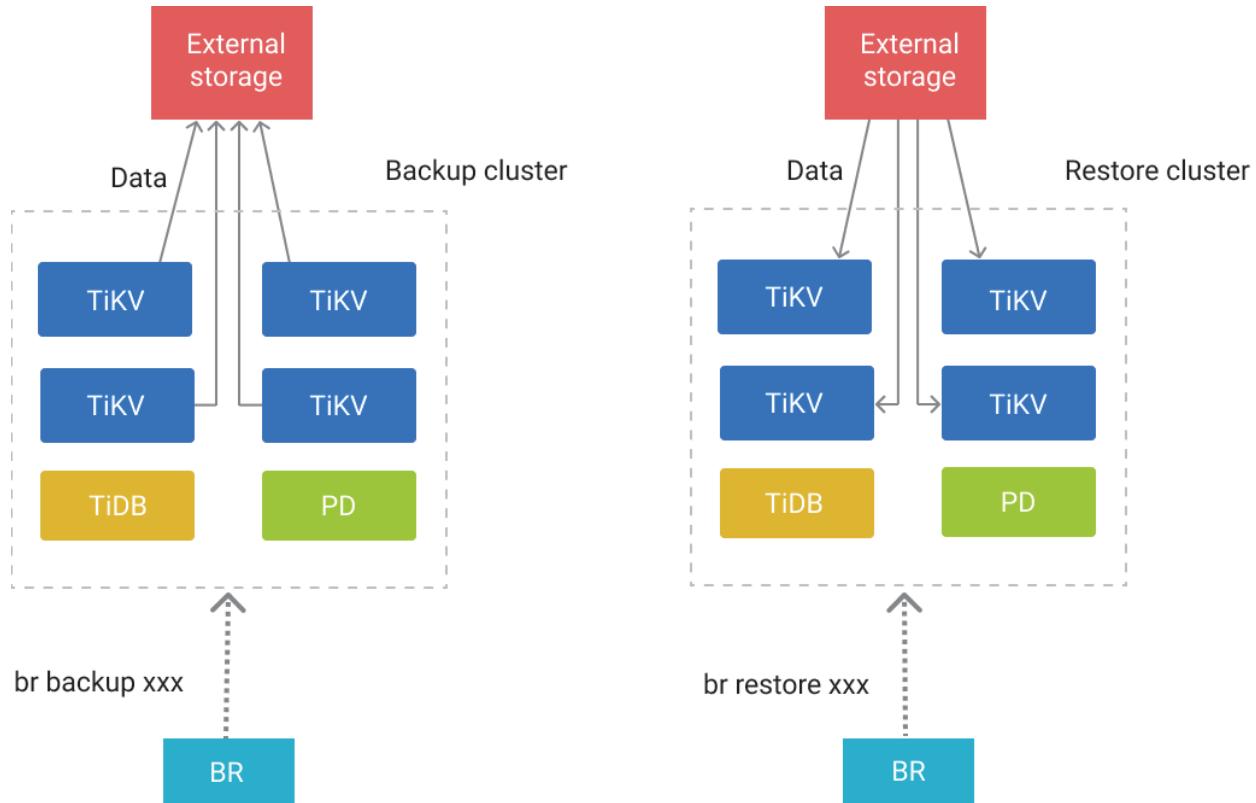


Figure 119: br-arch

### Backup principle

When BR performs a backup operation, it first obtains the following information from PD:

- The current TS (timestamp) as the time of the backup snapshot
- The TiKV node information of the current cluster

According to these information, BR starts a TiDB instance internally to obtain the database or table information corresponding to the TS, and filters out the system databases (`information_schema`, `performance_schema`, `mysql`) at the same time.

According to the backup sub-command, BR adopts the following two types of backup logic:

- Full backup: BR traverses all the tables and constructs the KV range to be backed up according to each table.
- Single table backup: BR constructs the KV range to be backed up according a single table.

Finally, BR collects the KV range to be backed up and sends the complete backup request to the TiKV node of the cluster.

The structure of the request:

```
BackupRequest{
    ClusterId,      // The cluster ID.
    StartKey,       // The starting key of the backup (backed up).
    EndKey,         // The ending key of the backup (not backed up).
    StartVersion,   // The version of the last backup snapshot, used for the
                    // → incremental backup.
    EndVersion,     // The backup snapshot time.
    StorageBackend, // The path where backup files are stored.
    RateLimit,      // Backup speed (MB/s).
}
```

After receiving the backup request, the TiKV node traverses all Region leaders on the node to find the Regions that overlap with the KV ranges in this request. The TiKV node backs up some or all of the data within the range, and generates the corresponding SST file.

After finishing backing up the data of the corresponding Region, the TiKV node returns the metadata to BR. BR collects the metadata and stores it in the `backupmeta` file which is used for restoration.

If `StartVersion` is not 0, the backup is seen as an incremental backup. In addition to KVs, BR also collects DDLs between `[StartVersion, EndVersion)`. During data restoration, these DDLs are restored first.

If checksum is enabled when you execute the backup command, BR calculates the checksum of each backed up table for data check.

#### 10.6.1.1.1 Types of backup files

Two types of backup files are generated in the path where backup files are stored:

- **The SST file:** stores the data that the TiKV node backed up.
- **The `backupmeta` file:** stores the metadata of this backup operation, including the number, the key range, the size, and the Hash (sha256) value of the backup files.
- **The `backup.lock` file:** prevents multiple backup operations from storing data to the same directory.

#### 10.6.1.1.2 The format of the SST file name

The SST file is named in the format of `storeID_regionID_regionEpoch_keyHash_cf`, where

- `storeID` is the TiKV node ID;
- `regionID` is the Region ID;

- `regionEpoch` is the version number of the Region;
- `keyHash` is the Hash (sha256) value of the `startKey` of a range, which ensures the uniqueness of a key;
- `cf` indicates the **Column Family** of RocksDB (`default` or `write` by default).

Restoration principle

During the data restoration process, BR performs the following tasks in order:

1. It parses the `backupmeta` file in the backup path, and then starts a TiDB instance internally to create the corresponding databases and tables based on the parsed information.
2. It aggregates the parsed SST files according to the tables.
3. It pre-splits Regions according to the key range of the SST file so that every Region corresponds to at least one SST file.
4. It traverses each table to be restored and the SST file corresponding to each tables.
5. It finds the Region corresponding to the SST file and sends a request to the corresponding TiKV node for downloading the file. Then it sends a request for loading the file after the file is successfully downloaded.

After TiKV receives the request to load the SST file, TiKV uses the Raft mechanism to ensure the strong consistency of the SST data. After the downloaded SST file is loaded successfully, the file is deleted asynchronously.

After the restoration operation is completed, BR performs a checksum calculation on the restored data to compare the stored data with the backed up data.

### 10.6.1.2 Deploy and use BR

#### 10.6.1.2.1 Recommended deployment configuration

- It is recommended that you deploy BR on the PD node.
- It is recommended that you mount a high-performance SSD to BR nodes and all TiKV nodes. A 10-gigabit network card is recommended. Otherwise, bandwidth is likely to be the performance bottleneck during the backup and restore process.

**Note:**

- If you do not mount a network disk or use other shared storage, the data backed up by BR will be generated on each TiKV node. Because BR only backs up leader replicas, you should estimate the space reserved for each node based on the leader size.
- Because TiDB uses leader count for load balancing by default, leaders can greatly differ in size. This might result in uneven distribution of backup data on each node.

#### 10.6.1.2.2 Usage restrictions

The following are the limitations of using BR for backup and restoration:

- When BR restores data to the upstream cluster of TiCDC/Drainer, TiCDC/Drainer cannot replicate the restored data to the downstream.
- BR supports operations only between clusters with the same `new_collations_enabled_on_first_backup` value because BR only backs up KV data. If the cluster to be backed up and the cluster to be restored use different collations, the data validation fails. Therefore, before restoring a cluster, make sure that the switch value from the query result of the `select VARIABLE_VALUE from mysql.tidb where VARIABLE_NAME='new_collation_enabled'`; statement is consistent with that during the backup process.

#### 10.6.1.2.3 Compatibility

The compatibility issues of BR and the TiDB cluster are divided into the following categories:

- Some versions of BR are not compatible with the interface of the TiDB cluster.
- The KV format might change when some features are enabled or disabled. If these features are not consistently enabled or disabled during backup and restore, compatibility issues might occur.

These features are as follows:



---

Related is- Features	Solutions
Related is- Features	Solutions
Clustered	#1565
index	Make sure that the value of the <code>tidb_enable_clustered_index</code> → global vari- able dur- ing re- store is consis- tent with that dur- ing backup. Other- wise, data incon- sis- tency might occur, such as <code>default</code> → → <code>not</code> → → <code>found</code> → 908 and incon- sis- tent

---

Features	Issues	Solutions
New collation	#352	<p>Make sure that the value of the <code>new_collations_enabled_on_first_bootstrap</code> variable is consistent with that during backup.</p> <p>Otherwise, inconsistent data index might occur and checksum might fail to pass.</p>

---

Feature	Issues	Solutions
TiCDC <a href="#">#364</a>	Currently, TiKV can not push down the BR-ingested SST files to TiCDC. Therefore, you need to disable TiCDC when using BR to re-store data.	

Features	Issues	Solutions
Global temporary tables	Make sure that you are using BR v5.3.0 or a later version to back up and restore data. Otherwise, an error occurs in the definition of the backed global temporary tables.	Related issues

However, even after you have ensured that the above features are consistently enabled or disabled during backup and restore, compatibility issues might still occur due to the inconsistent internal versions or inconsistent interfaces between BR and TiKV/TiDB/PD. To avoid such cases, BR has the built-in version check.

## Version check

Before performing backup and restore, BR compares and checks the TiDB cluster version and the BR version. If there is a major-version mismatch (for example, BR v4.x and TiDB v5.x), BR prompts a reminder to exit. To forcibly skip the version check, you can set `--check-requirements=false`.

Note that skipping the version check might introduce incompatibility. The version compatibility information between BR and TiDB versions are as follows:

---

Backup			
ver-			
sion			
(ver-			
tical)			
Re-	Use	Use	Use
store	BR	BR	BR
ver-	nightly	v5.0	v4.0
sion	to re-	to re-	to re-
(hori-	store	store	store
zon-	TiDB	TiDB	TiDB
tal)	nightly	v5.0	v4.0
Use			(If
BR			a
nightly			table
to			with
back			the
up			pri-
TiDB			mmary
nightly			key
			of
			the
			non-
			integer
			clus-
			tered
			index
			type
			is re-
			stored
			to a
			TiDB
			v4.0
			clus-
			ter,
			BR
			will
			cause
			data
			error
			with-
			out
			warn-
			ing.)

---

Backup

ver-

sion

(ver-

tical)

Re-	Use	Use	Use
store	BR	BR	BR
ver-	nightly	v5.0	v4.0
sion	to re-	to re-	to re-
(hori-	store	store	store
zon-	TiDB	TiDB	TiDB
tal)	nightly	v5.0	v4.0

---

Use (If  
BR a  
v5.0 table  
to with  
back the  
up pri-  
TiDB mary  
v5.0 key  
of  
the  
non-  
integer  
clus-  
tered  
index  
type  
is re-  
stored  
to a  
TiDB  
v4.0  
clus-  
ter,  
BR  
will  
cause  
data  
error  
with-  
out  
warn-  
ing.)

## Backup

ver-

sion

(ver-

tical)

Re-	Use	Use	Use
store	BR	BR	BR
ver-	nightly	v5.0	v4.0
sion	to re-	to re-	to re-
(hori-	store	store	store
zon-	TiDB	TiDB	TiDB
tal)	nightly	v5.0	v4.0

Use (If  
BR TiKV  
v4.0 >= v4.0.0-  
to v4.0.0-  
back rc.1,  
up and  
TiDB if BR  
v4.0 contains  
the  
[#233](#)  
bug  
fix  
and  
TiKV  
does  
not  
con-  
tain  
the  
[#7241](#)  
bug  
fix,  
BR  
will  
cause  
the  
TiKV  
node  
to  
restart

---

Backup			
ver-			
sion			
(ver-			
tical)			
Re-	Use	Use	Use
store	BR	BR	BR
ver-	nightly	v5.0	v4.0
sion	to re-	to re-	to re-
(hori-	store	store	store
zon-	TiDB	TiDB	TiDB
tal)	nightly	v5.0	v4.0
Use	(If	(If	(If
BR	the	the	the
nightly	TiDB	TiDB	TiDB
or	ver-	ver-	ver-
v5.0	sion	sion	sion
to	is	is	is
back	ear-	ear-	ear-
up	lier	lier	lier
TiDB	than	than	than
v4.0	v4.0.9,	v4.0.9,	v4.0.9,
	the	the	the
	#609	#609	#609
	issue	issue	issue
	might	might	might
	oc-	oc-	oc-
	cur.)	cur.)	cur.)

---

#### 10.6.1.2.4 Back up and restore table data in the `mysql` system schema (experimental feature)

**Warning:**

This feature is experimental and not thoroughly tested. It is highly **not recommended** to use this feature in the production environment.

Before v5.1.0, BR filtered out data from the system schema `mysql` during the backup. Since v5.1.0, BR **backs up** all data by default, including the system schemas `mysql.*`. But the technical implementation of restoring the system tables in `mysql.*` is not complete yet, so the tables in the system schema `mysql` are **not** restored by default.

If you want the data of a system table (for example, `mysql.usertable1`) to be restored to the system schema `mysql`, you can set the **filter parameter** to filter the table name (`-r f "mysql.usertable1"`). After the setting, the system table is first restored to the temporary schema, and then to the system schema through renaming.

It should be noted that the following system tables cannot be restored correctly due to technical reasons. Even if `-f "mysql.*"` is specified, these tables will not be restored:

- Tables related to statistics: “`stats_buckets`”, “`stats_extended`”, “`stats_feedback`”, “`stats_fm_sketch`”, “`stats_histograms`”, “`stats_meta`”, “`stats_top_n`”
- Tables related to privileges or the system: “`tidb`”, “`global_variables`”, “`columns_priv`”, “`db`”, “`default_roles`”, “`global_grants`”, “`global_priv`”, “`role_edges`”, “`tables_priv`”, “`user`”, “`gc_delete_range`”, “`Gc_delete_range_done`”, “`schema_index_usage`”

#### 10.6.1.2.5 Minimum machine configuration required for running BR

The minimum machine configuration required for running BR is as follows:

CPU	Memory	Hard Disk Type	Network
1 core	4 GB	HDD	Gigabit network card

In general scenarios (less than 1000 tables for backup and restore), the CPU consumption of BR at runtime does not exceed 200%, and the memory consumption does not exceed 4 GB. However, when backing up and restoring a large number of tables, BR might consume more than 4 GB of memory. In a test of backing up 24000 tables, BR consumes about 2.7 GB of memory, and the CPU consumption remains below 100%.

#### 10.6.1.2.6 Best practices

The following are some recommended operations for using BR for backup and restoration:

- It is recommended that you perform the backup operation during off-peak hours to minimize the impact on applications.
- BR supports restore on clusters of different topologies. However, the online applications will be greatly impacted during the restore operation. It is recommended that you perform restore during the off-peak hours or use `rate-limit` to limit the rate.
- It is recommended that you execute multiple backup operations serially. Running different backup operations in parallel reduces backup performance and also affects the online application.
- It is recommended that you execute multiple restore operations serially. Running different restore operations in parallel increases Region conflicts and also reduces restore performance.
- It is recommended that you mount a shared storage (for example, NFS) on the backup path specified by `-s`, to make it easier to collect and manage backup files.

- It is recommended that you use a storage hardware with high throughput, because the throughput of a storage hardware limits the backup and restoration speed.
- It is recommended that you disable the checksum feature (`--checksum = false`) during backup operation and only enable it during the restore operation to reduce migration time. This is because BR by default respectively performs checksum calculation after backup and restore operations to compare the stored data with the corresponding cluster data to ensure accuracy.

#### 10.6.1.2.7 How to use BR

Currently, the following methods are supported to run the BR tool:

- Use SQL statements
- Use the command-line tool
- Use BR In the Kubernetes environment

Use SQL statements

TiDB supports both `BACKUP` and `RESTORE` SQL statements. The progress of these operations can be monitored with the statement `SHOW BACKUPS | RESTORES`.

Use the command-line tool

The `br` command-line utility is available as a [separate download](#). For details, see [Use BR Command-line for Backup and Restoration](#).

In the Kubernetes environment

In the Kubernetes environment, you can use the BR tool to back up TiDB cluster data to S3-compatible storage, Google Cloud Storage (GCS) and persistent volumes (PV), and restore them:

**Note:**

For Amazon S3 and Google Cloud Storage parameter descriptions, see the [External Storages](#) document.

- Back up Data to S3-Compatible Storage Using BR
- Restore Data from S3-Compatible Storage Using BR
- Back up Data to GCS Using BR
- Restore Data from GCS Using BR
- Back up Data to PV Using BR
- Restore Data from PV Using BR

### 10.6.1.3 Other documents about BR

- Use BR Command-line
- BR Use Cases
- BR FAQ
- External Storages

### 10.6.2 Use BR Command-line for Backup and Restoration

This document describes how to back up and restore TiDB cluster data using the BR command line.

Make sure you have read [BR Tool Overview](#), especially [Usage Restrictions](#) and [Best Practices](#).

#### 10.6.2.1 BR command-line description

A `br` command consists of sub-commands, options, and parameters.

- Sub-command: the characters without `-` or `--`.
- Option: the characters that start with `-` or `--`.
- Parameter: the characters that immediately follow behind and are passed to the sub-command or the option.

This is a complete `br` command:

```
br backup full --pd "${PDIP}:2379" -s "local:///tmp/backup"
```

Explanations for the above command are as follows:

- `backup`: the sub-command of `br`.
- `full`: the sub-command of `backup`.
- `-s` (or `--storage`): the option that specifies the path where the backup files are stored.
- `"local:///tmp/backup"`: the parameter of `-s`. `/tmp/backup` is the path in the local disk where the backed up files of each TiKV node are stored.
- `--pd`: the option that specifies the Placement Driver (PD) service address.
- `"${PDIP}:2379"`: the parameter of `--pd`.

#### Note:

- When the `local` storage is used, the backup data are scattered in the local file system of each node.

- It is **not recommended** to back up to a local disk in the production environment because you **have to** manually aggregate these data to complete the data restoration. For more information, see [Restore Cluster Data](#).
- Aggregating these backup data might cause redundancy and bring troubles to operation and maintenance. Even worse, if restoring data without aggregating these data, you can receive a rather confusing error message **SST file not found**.
- It is recommended to mount the NFS disk on each node, or back up to the **S3** object storage.

#### 10.6.2.1.1 Sub-commands

A **br** command consists of multiple layers of sub-commands. Currently, BR has the following three sub-commands:

- **br backup**: used to back up the data of the TiDB cluster.
- **br restore**: used to restore the data of the TiDB cluster.

Each of the above three sub-commands might still include the following three sub-commands to specify the scope of an operation:

- **full**: used to back up or restore all the cluster data.
- **db**: used to back up or restore the specified database of the cluster.
- **table**: used to back up or restore a single table in the specified database of the cluster.

#### 10.6.2.1.2 Common options

- **--pd**: used for connection, specifying the PD server address. For example, "`-${PDIP} ↵ }:2379`".
- **-h** (or **--help**): used to get help on all sub-commands. For example, `br backup -- ↵ help`.
- **-V** (or **--version**): used to check the version of BR.
- **--ca**: specifies the path to the trusted CA certificate in the PEM format.
- **--cert**: specifies the path to the SSL certificate in the PEM format.
- **--key**: specifies the path to the SSL certificate key in the PEM format.
- **--status-addr**: specifies the listening address through which BR provides statistics to Prometheus.

### 10.6.2.2 Use BR command-line to back up cluster data

To back up the cluster data, use the `br backup` command. You can add the `full` or `table` sub-command to specify the scope of your backup operation: the whole cluster or a single table.

#### 10.6.2.2.1 Back up all the cluster data

To back up all the cluster data, execute the `br backup full` command. To get help on this command, execute `br backup full -h` or `br backup full --help`.

##### Usage example:

Back up all the cluster data to the `/tmp/backup` path of each TiKV node and write the `backupmeta` file to this path.

##### Note:

- If the backup disk and the service disk are different, it has been tested that online backup reduces QPS of the read-only online service by about 15%-25% in case of full-speed backup. If you want to reduce the impact on QPS, use `--ratelimit` to limit the rate.
- If the backup disk and the service disk are the same, the backup competes with the service for I/O resources. This might decrease the QPS of the read-only online service by more than half. Therefore, it is **highly not recommended** to back up the online service data to the TiKV data disk.

```
br backup full \
  --pd "${PDIP}:2379" \
  --storage "local:///tmp/backup" \
  --ratelimit 128 \
  --log-file backupfull.log
```

Explanations for some options in the above command are as follows:

- `--ratelimit`: specifies the maximum speed at which a backup operation is performed (MiB/s) on each TiKV node.
- `--log-file`: specifies writing the BR log to the `backupfull.log` file.

A progress bar is displayed in the terminal during the backup. When the progress bar advances to 100%, the backup is complete. Then the BR also checks the backup data to ensure data safety. The progress bar is displayed as follows:

```
br backup full \
--pd "${PDIP}:2379" \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backupfull.log
Full Backup <-----/.....>
↪ 17.12%.
```

#### 10.6.2.2.2 Back up a database

To back up a database in the cluster, execute the `br backup db` command. To get help on this command, execute `br backup db -h` or `br backup db --help`.

##### Usage example:

Back up the data of the `test` database to the `/tmp/backup` path on each TiKV node and write the `backupmeta` file to this path.

```
br backup db \
--pd "${PDIP}:2379" \
--db test \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backuptable.log
```

In the above command, `--db` specifies the name of the database to be backed up. For descriptions of other options, see [Back up all the cluster data](#).

A progress bar is displayed in the terminal during the backup. When the progress bar advances to 100%, the backup is complete. Then the BR also checks the backup data to ensure data safety.

#### 10.6.2.2.3 Back up a table

To back up the data of a single table in the cluster, execute the `br backup table` command. To get help on this command, execute `br backup table -h` or `br backup ↪ table --help`.

##### Usage example:

Back up the data of the `test.usertable` table to the `/tmp/backup` path on each TiKV node and write the `backupmeta` file to this path.

```
br backup table \
--pd "${PDIP}:2379" \
--db test \
--table usertable \
--storage "local:///tmp/backup" \
```

```
--ratelimit 128 \
--log-file backuptable.log
```

The `table` sub-command has two options:

- `--db`: specifies the database name
- `--table`: specifies the table name.

For descriptions of other options, see [Back up all cluster data](#).

A progress bar is displayed in the terminal during the backup operation. When the progress bar advances to 100%, the backup is complete. Then the BR also checks the backup data to ensure data safety.

#### 10.6.2.2.4 Back up with table filter

To back up multiple tables with more complex criteria, execute the `br backup full` command and specify the `table filters` with `--filter` or `-f`.

##### Usage example:

The following command backs up the data of all tables in the form `db*.tbl*` to the `/tmp/backup` path on each TiKV node and writes the `backupmeta` file to this path.

```
br backup full \
--pd "${PDIP}:2379" \
--filter 'db*.tbl*' \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file backupfull.log
```

#### 10.6.2.2.5 Back up data to Amazon S3 backend

If you back up the data to the Amazon S3 backend, instead of `local` storage, you need to specify the S3 storage path in the `storage` sub-command, and allow the BR node and the TiKV node to access Amazon S3.

You can refer to the [AWS Official Document](#) to create an S3 Bucket in the specified Region. You can also refer to another [AWS Official Document](#) to create a Folder in the Bucket.

##### Note:

To complete one backup, TiKV and BR usually require the minimum privileges of `s3>ListBucket`, `s3:PutObject`, and `s3:AbortMultipartUpload`.

Pass `SecretKey` and `AccessKey` of the account that has privilege to access the S3 backend to the BR node. Here `SecretKey` and `AccessKey` are passed as environment variables. Then pass the privilege to the TiKV node through BR.

```
export AWS_ACCESS_KEY_ID=${AccessKey}
export AWS_SECRET_ACCESS_KEY=${SecretKey}
```

When backing up using BR, explicitly specify the parameters `--s3.region` and `--send-credentials-to-tikv`. `--s3.region` indicates the region where S3 is located, and `--send-credentials-to-tikv` means passing the privilege to access S3 to the TiKV node.

```
br backup full \
  --pd "${PDIP}:2379" \
  --storage "s3://${Bucket}/${Folder}" \
  --s3.region "${region}" \
  --send-credentials-to-tikv=true \
  --ratelimit 128 \
  --log-file backuptable.log
```

#### 10.6.2.2.6 Back up incremental data

If you want to back up incrementally, you only need to specify the `last backup timestamp` `--lastbackups`.

The incremental backup has two limitations:

- The incremental backup needs to be under a different path from the previous full backup.
- GC (Garbage Collection) safepoint must be before the `lastbackups`.

To back up the incremental data between `(LAST_BACKUP_TS, current PD timestamp]`, execute the following command:

```
br backup full\
  --pd ${PDIP}:2379 \
  --ratelimit 128 \
  -s local:///home/tidb/backupdata/incr \
  --lastbackups ${LAST_BACKUP_TS}
```

To get the timestamp of the last backup, execute the `validate` command. For example:

```
LAST_BACKUP_TS=`br validate decode --field="end-version" -s local:///home/
  ↪ tidb/backupdata | tail -n1`
```

In the above example, for the incremental backup data, BR records the data changes and the DDL operations during `(LAST_BACKUP_TS, current PD timestamp]`. When restoring data, BR first restores DDL operations and then the data.

#### 10.6.2.2.7 Encrypt data during backup (experimental feature)

Since TiDB v5.3.0, TiDB supports backup encryption. You can configure the following parameters to encrypt data during backup:

- `--crypter.method`: Encryption algorithm. Supports three algorithms `aes128-ctr`/  
↪ `aes192-ctr/aes256-ctr`. The default value is `plaintext` and indicates no encryption.
- `--crypter.key`: Encryption key in hexadecimal string format. `aes128-ctr` means 128 bit (16 bytes) key length, `aes192-ctr` means 24 bytes and `aes256-ctr` means 32 bytes.
- `--crypter.key-file`: The key file. You can directly pass in the file path where the key is stored as a parameter without passing in “`crypter.key`”

**Warning:**

- This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.
- If the key is lost, the backup data cannot be restored to the cluster.
- The encryption feature needs to be used on BR tools and TiDB clusters v5.3.0 or later versions, and the encrypted backup data cannot be restored on clusters earlier than v5.3.0.

The configuration example for backup encryption is as follows:

```
br backup full\
  --pd ${PDIP}:2379 \
  -s local:///home/tidb/backupdata/incr \
  --crypter.method aes128-ctr \
  --crypter.key 0123456789abcdef0123456789abcdef
```

#### 10.6.2.2.8 Point-in-time recovery (experimental feature)

Point-in-time recovery (PITR) allows you to restore data to a point in time of your choice.

An example scenario would be to take a full backup every day and take incremental backups every 6 hours and then use TiCDC for PITR. Assume that on one day, the full backup was performed at 00:00 and the first incremental backup was performed at 06:00. If you want to restore the database to the state of 07:16, you can first restore the full backup (taken at 00:00) and the incremental backup (taken at 06:00), and then restore TiCDC logs that fill in the gap between 06:00 and 07:16.

To perform the PITR, you can take the following steps:

1. Restore a full backup using `br restore full`.
2. (optional) Restore incremental backup(s).
3. Use `br restore cdclog` to restore the transactions that happened after the last incremental backup. The complete command to execute is as follows:

```
br restore cdclog --storage local:///data/cdclog --start-ts $START_TS
    ↪ --end-ts $END_TS
```

In the command above:

- `local:///data/cdclog` is the location of the TiCDC logs. This might be on the local filesystem or on the external storage like S3.
- `$START_TS` is the end position of the restore from the last restored backup (either a full backup or an incremental backup).
- `$END_TS` is the point to which you want to restore your data.

#### 10.6.2.2.9 Back up Raw KV (experimental feature)

**Warning:**

This feature is experimental and not thoroughly tested. It is highly **not recommended** to use this feature in the production environment.

In some scenarios, TiKV might run independently of TiDB. Given that, BR also supports bypassing the TiDB layer and backing up data in TiKV.

For example, you can execute the following command to back up all keys between [0x31, 0x3130303030303030] in the default CF to `$BACKUP_DIR`:

```
br backup raw --pd $PD_ADDR \
    -s "local://$BACKUP_DIR" \
    --start 31 \
    --ratelimit 128 \
    --end 3130303030303030 \
    --format hex \
    --cf default
```

Here, the parameters of `--start` and `--end` are decoded using the method specified by `--format` before being sent to TiKV. Currently, the following methods are available:

- “raw”: The input string is directly encoded as a key in binary format.
- “hex”: The default encoding method. The input string is treated as a hexadecimal number.
- “escape”: First escape the input string, and then encode it into binary format.

### 10.6.2.3 Use BR command-line to restore cluster data

To restore the cluster data, use the `br restore` command. You can add the `full`, `db` or `table` sub-command to specify the scope of your restoration: the whole cluster, a database or a single table.

#### Note:

If you use the local storage, you **must** copy all back up SST files to every TiKV node in the path specified by `--storage`.

Even if each TiKV node eventually only need to read a part of the all SST files, they all need full access to the complete archive because:

- Data are replicated into multiple peers. When ingesting SSTs, these files have to be present on *all* peers. This is unlike back up where reading from a single node is enough.
- Where each peer is scattered to during restore is random. We don't know in advance which node will read which file.

These can be avoided using shared storage, for example mounting an NFS on the local path, or using S3. With network storage, every node can automatically read every SST file, so these caveats no longer apply.

Also, note that you can only run one restore operation for a single cluster at the same time. Otherwise, unexpected behaviors might occur. For details, see [FAQ](#).

#### 10.6.2.3.1 Restore all the backup data

To restore all the backup data to the cluster, execute the `br restore full` command. To get help on this command, execute `br restore full -h` or `br restore full --help`.

#### Usage example:

Restore all the backup data in the `/tmp/backup` path to the cluster.

```
br restore full \
--pd "${PDIP}:2379" \
--storage "local:///tmp/backup" \
--ratelimit 128 \
--log-file restorefull.log
```

Explanations for some options in the above command are as follows:

- `--ratelimit`: specifies the maximum speed at which a restoration operation is performed (MiB/s) on each TiKV node.

- `--log-file`: specifies writing the BR log to the `restorefull.log` file.

A progress bar is displayed in the terminal during the restoration. When the progress bar advances to 100%, the restoration is complete. Then the BR also checks the backup data to ensure data safety.

```
br restore full \
  --pd "${PDIP}:2379" \
  --storage "local:///tmp/backup" \
  --ratelimit 128 \
  --log-file restorefull.log
Full Restore <-----/.....>
→ 17.12%.
```

#### 10.6.2.3.2 Restore a database

To restore a database to the cluster, execute the `br restore db` command. To get help on this command, execute `br restore db -h` or `br restore db --help`.

##### Usage example:

Restore a database backed up in the `/tmp/backup` path to the cluster.

```
br restore db \
  --pd "${PDIP}:2379" \
  --db "test" \
  --ratelimit 128 \
  --storage "local:///tmp/backup" \
  --log-file restorefull.log
```

In the above command, `--db` specifies the name of the database to be restored. For descriptions of other options, see [Restore all backup data](#)).

##### Note:

When you restore the backup data, the name of the database specified by `--db` must be the same as the one specified by `--db` in the backup command. Otherwise, the restore fails. This is because the metafile of the backup data (`backupmeta` file) records the database name, you can only restore data to the database with the same name. The recommended method is to restore the backup data to the database with the same name in another cluster.

#### 10.6.2.3.3 Restore a table

To restore a single table to the cluster, execute the `br restore table` command. To get help on this command, execute `br restore table -h` or `br restore table --help`.

##### Usage example:

Restore a table backed up in the `/tmp/backup` path to the cluster.

```
br restore table \
--pd "${PDIP}:2379" \
--db "test" \
--table "usertable" \
--ratelimit 128 \
--storage "local:///tmp/backup" \
--log-file restorefull.log
```

In the above command, `--table` specifies the name of the table to be restored. For descriptions of other options, see [Restore all backup data](#) and [Restore a database](#).

#### 10.6.2.3.4 Restore with table filter

To restore multiple tables with more complex criteria, execute the `br restore full` command and specify the `table filters` with `--filter` or `-f`.

##### Usage example:

The following command restores a subset of tables backed up in the `/tmp/backup` path to the cluster.

```
br restore full \
--pd "${PDIP}:2379" \
--filter 'db*.tbl*' \
--storage "local:///tmp/backup" \
--log-file restorefull.log
```

#### 10.6.2.3.5 Restore data from Amazon S3 backend

If you restore data from the Amazon S3 backend, instead of `local` storage, you need to specify the S3 storage path in the `storage` sub-command, and allow the BR node and the TiKV node to access Amazon S3.

##### Note:

To complete one restore, TiKV and BR usually require the minimum privileges of `s3>ListBucket` and `s3GetObject`.

Pass `SecretKey` and `AccessKey` of the account that has privilege to access the S3 backend to the BR node. Here `SecretKey` and `AccessKey` are passed as environment variables. Then pass the privilege to the TiKV node through BR.

```
export AWS_ACCESS_KEY_ID=${AccessKey}
export AWS_SECRET_ACCESS_KEY=${SecretKey}
```

When restoring data using BR, explicitly specify the parameters `--s3.region` and `--send-credentials-to-tikv`. `--s3.region` indicates the region where S3 is located, and `--send-credentials-to-tikv` means passing the privilege to access S3 to the TiKV node.

`Bucket` and `Folder` in the `--storage` parameter represent the S3 bucket and the folder where the data to be restored is located.

```
br restore full \
  --pd "${PDIP}:2379" \
  --storage "s3://${Bucket}/${Folder}" \
  --s3.region "${region}" \
  --ratelimit 128 \
  --send-credentials-to-tikv=true \
  --log-file restorefull.log
```

In the above command, `--table` specifies the name of the table to be restored. For descriptions of other options, see [Restore a database](#).

#### 10.6.2.3.6 Restore incremental data

Restoring incremental data is similar to [restoring full data using BR](#). Note that when restoring incremental data, make sure that all the data backed up before `last backup ts` has been restored to the target cluster.

#### 10.6.2.3.7 Restore tables created in the mysql schema (experimental feature)

BR backs up tables created in the `mysql` schema by default.

When you restore data using BR, the tables created in the `mysql` schema are not restored by default. If you need to restore these tables, you can explicitly include them using the `table filter`. The following example restores `mysql.usertable` created in `mysql` schema. The command restores `mysql.usertable` along with other data.

```
br restore full -f '*.*' -f '!mysql.*' -f 'mysql.usertable' -s
  ↪ $external_storage_url --ratelimit 128
```

In the above command, `-f '*.*'` is used to override the default rules and `-f '!mysql.*'` instructs BR not to restore tables in `mysql` unless otherwise stated. `-f 'mysql.usertable'` indicates that `mysql.usertable` is required for restore. For detailed implementation, refer to the [table filter document](#).

If you only need to restore `mysql.usertable`, use the following command:

```
br restore full -f 'mysql.usertable' -s $external_storage_url --ratelimit
↪ 128
```

### **Warning:**

Although you can back up and restore system tables (such as `mysql.tidb`) using the BR tool, some unexpected situations might occur after the restore, including:

- the statistical information tables (`mysql.stat_*`) cannot be restored.
- the system variable tables (`mysql.tidb`,`mysql.global_variables`) cannot be restored.
- the user information tables (such as `mysql.user` and `mysql.
↪ columns_priv`) cannot be restored.
- GC data cannot be restored.

Restoring system tables might cause more compatibility issues. To avoid unexpected issues, **DO NOT** restore system tables in the production environment.

### 10.6.2.3.8 Decrypt data during restore (experimental feature)

### **Warning:**

This is still an experimental feature. It is **NOT** recommended that you use it in the production environment.

After encrypting the backup data, you need to pass in the corresponding decryption parameters to restore the data. You need to ensure that the decryption parameters and encryption parameters are consistent. If the decryption algorithm or key is incorrect, the data cannot be restored.

The following is an example of decrypting the backup data:

```
br restore full\
--pd ${PDIP}:2379 \
-s local:///home/tidb/backupdata/incr \
--crypter.method aes128-ctr \
--crypter.key 0123456789abcdef0123456789abcdef
```

### 10.6.2.3.9 Restore Raw KV (experimental feature)

**Warning:**

This feature is in the experiment, without being thoroughly tested. It is highly **not recommended** to use this feature in the production environment.

Similar to [backing up Raw KV](#), you can execute the following command to restore Raw KV:

```
br restore raw --pd $PD_ADDR \
-s "local://$BACKUP_DIR" \
--start 31 \
--end 3130303030303030 \
--ratelimit 128 \
--format hex \
--cf default
```

In the above example, all the backed up keys in the range [0x31, 0x3130303030303030 ↵) are restored to the TiKV cluster. The coding methods of these keys are identical to that of [keys during the backup process](#)

### 10.6.2.3.10 Online restore (experimental feature)

**Warning:**

This feature is in the experiment, without being thoroughly tested. It also relies on the unstable [Placement Rules](#) feature of PD. It is highly **not recommended** to use this feature in the production environment.

During data restoration, writing too much data affects the performance of the online cluster. To avoid this effect as much as possible, BR supports [Placement rules](#) to isolate resources. In this case, downloading and importing SST are only performed on a few specified nodes (or “restore nodes” for short). To complete the online restore, take the following steps.

1. Configure PD, and start Placement rules:

```
echo "config set enable-placement-rules true" | pd-ctl
```

2. Edit the configuration file of the “restore node” in TiKV, and specify “restore” to the `server` configuration item:

```
[server]
labels = { exclusive = "restore" }
```

3. Start TiKV of the “restore node” and restore the backed up files using BR. Compared with the offline restore, you only need to add the `--online` flag:

```
br restore full \
-s "local://$BACKUP_DIR" \
--ratelimit 128 \
--pd $PD_ADDR \
--online
```

### 10.6.3 BR Use Cases

**BR** is a tool for distributed backup and restoration of the TiDB cluster data.

This document describes how to run BR in the following use cases:

- Back up a single table to a network disk (recommended in production environment)
- Restore data from a network disk (recommended in production environment)
- Back up a single table to a local disk (recommended in testing environment)
- Restore data from a local disk (recommended in testing environment)

This document aims to help you achieve the following goals:

- Back up and restore data using a network disk or local disk correctly.
- Get the status of a backup or restoration operation through monitoring metrics.
- Learn how to tune performance during the operation.
- Troubleshoot the possible anomalies during the backup operation.

#### 10.6.3.1 Audience

You are expected to have a basic understanding of TiDB and [TiKV](#).

Before reading on, make sure you have read [BR Tool Overview](#), especially [Usage Restrictions](#) and [Best Practices](#).

#### 10.6.3.2 Prerequisites

This section introduces the recommended method of deploying TiDB, cluster versions, the hardware information of the TiKV cluster, and the cluster configuration for the use case demonstrations.

You can estimate the performance of your backup or restoration operation based on your own hardware and configuration.

#### 10.6.3.2.1 Deployment method

It is recommended that you deploy the TiDB cluster using [TiUP](#) and get BR by downloading [TiDB Toolkit](#).

#### 10.6.3.2.2 Cluster versions

- TiDB: v5.0.0
- TiKV: v5.0.0
- PD: v5.0.0
- BR: v5.0.0

##### Note:

v5.0.0 was the latest version at the time this document was written. It is recommended that you use the latest version of [TiDB/TiKV/PD/BR](#) and make sure that the BR version is **consistent with** the TiDB version.

#### 10.6.3.2.3 TiKV hardware information

- Operating system: CentOS Linux release 7.6.1810 (Core)
- CPU: 16-Core Common KVM processor
- RAM: 32GB
- Disk: 500G SSD \* 2
- NIC: 10 Gigabit network card

#### 10.6.3.2.4 Cluster configuration

BR directly sends commands to the TiKV cluster and are not dependent on the TiDB server, so you do not need to configure the TiDB server when using BR.

- TiKV: default configuration
- PD: default configuration

#### 10.6.3.3 Use cases

This document describes the following use cases:

- Back up a single table to a network disk (recommended in production environment)
- Restore data from a network disk (recommended in production environment)
- Back up a single table to a local disk (recommended in testing environment)

- **Restore data from a local disk (recommended in testing environment)**

It is recommended that you use a network disk to back up and restore data. This spares you from collecting backup files and greatly improves the backup efficiency especially when the TiKV cluster is in a large scale.

Before the backup or restoration operations, you need to do some preparations:

- Preparation for backup
- Preparation for restoration

#### 10.6.3.3.1 Preparation for backup

The BR tool already supports self-adapting to GC. It automatically registers `backupTS` (the latest PD timestamp by default) to PD's `safePoint` to ensure that TiDB's GC Safe Point does not move forward during the backup, thus avoiding manually setting GC configurations.

For the detailed usage of the `br backup` command, refer to [Use BR Command-line for Backup and Restoration](#).

1. Before executing the `br backup` command, ensure that no DDL is running on the TiDB cluster.
2. Ensure that the storage device where the backup will be created has sufficient space.

#### 10.6.3.3.2 Preparation for restoration

Before executing the `br restore` command, check the new cluster to make sure that the table in the cluster does not have a duplicate name.

#### 10.6.3.3.3 Back up a single table to a network disk (recommended in production environment)

Use the `br backup` command to back up the single table data `--db batchmark --table order_line` to the specified path `local:///br_data` in the network disk.

Backup prerequisites

- Preparation for backup
- Configure a high-performance SSD hard disk host as the NFS server to store data, and all BR nodes, TiKV nodes, and TiFlash nodes as NFS clients. Mount the same path (for example, `/br_data`) to the NFS server for NFS clients to access the server.
- The total transfer rate between the NFS server and all NFS clients must reach at least the number of TiKV instances \* 150MB/s. Otherwise the network I/O might become the performance bottleneck.

### Note:

- During data backup, because only the data of leader replicas are backed up, even if there is a TiFlash replica in the cluster, BR can complete the backup without mounting TiFlash nodes.
- When restoring data, BR will restore the data of all replicas. Also, TiFlash nodes need access to the backup data for BR to complete the restore. Therefore, before the restore, you must mount TiFlash nodes to the NFS server.

### Topology

The following diagram shows the typology of BR:

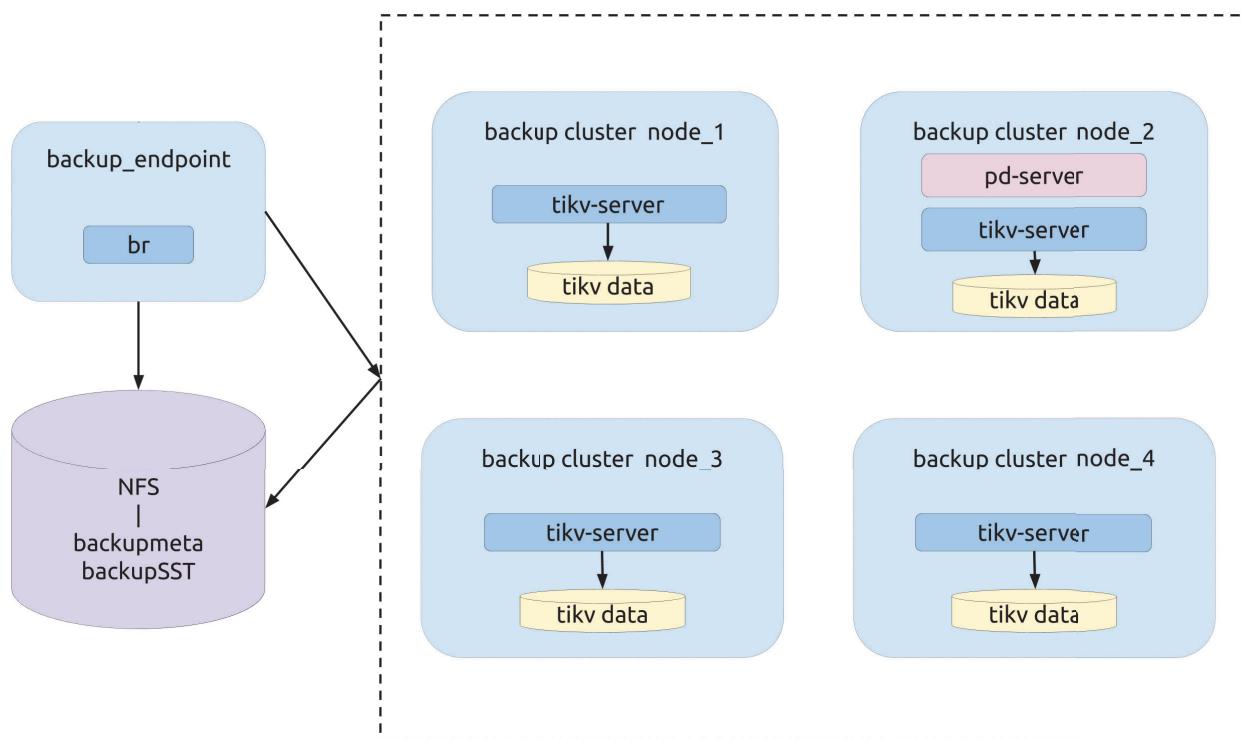


Figure 120: img

### Backup operation

Before the backup operation, execute the `admin checksum table order_line` command to get the statistical information of the table to be backed up (`--db batchmark` → `--table order_line`). The following image shows an example of this information:

Db_name	Table_name	Checksum_crc64_xor	Total_kvs	Total_bytes
batchmark	order_line	10912722838344822475	5659888624	370385538778
<b>1 row in set (5 min 47.59 sec)</b>				

Figure 121: img

Execute the `br backup` command:

```
bin/br backup table \
--db batchmark \
--table order_line \
-s local:///br_data \
--pd ${PD_ADDR}:2379 \
--log-file backup-nfs.log
```

Monitoring metrics for the backup

During the backup process, pay attention to the following metrics on the monitoring panels to get the status of the backup process.

**Backup CPU Utilization:** the CPU usage rate of each working TiKV node in the backup operation (for example, backup-worker and backup-endpoint).

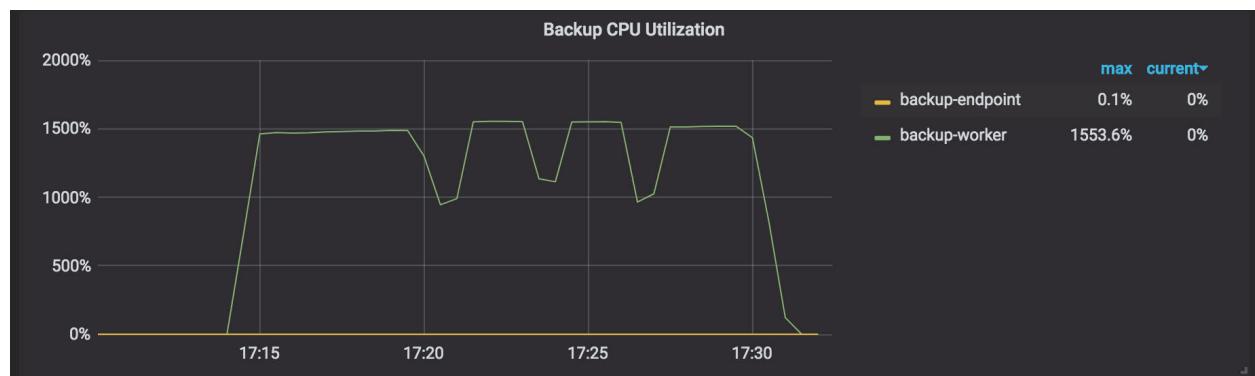


Figure 122: img

**IO Utilization:** the I/O usage rate of each working TiKV node in the backup operation.

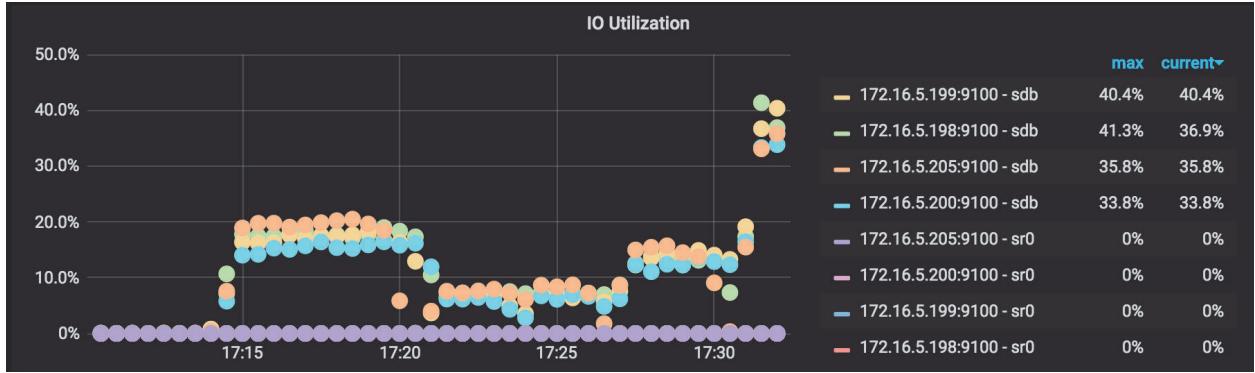


Figure 123: img

**BackupSST Generation Throughput:** the backupSST generation throughput of each working TiKV node in the backup operation, which is normally around 150MB/s.

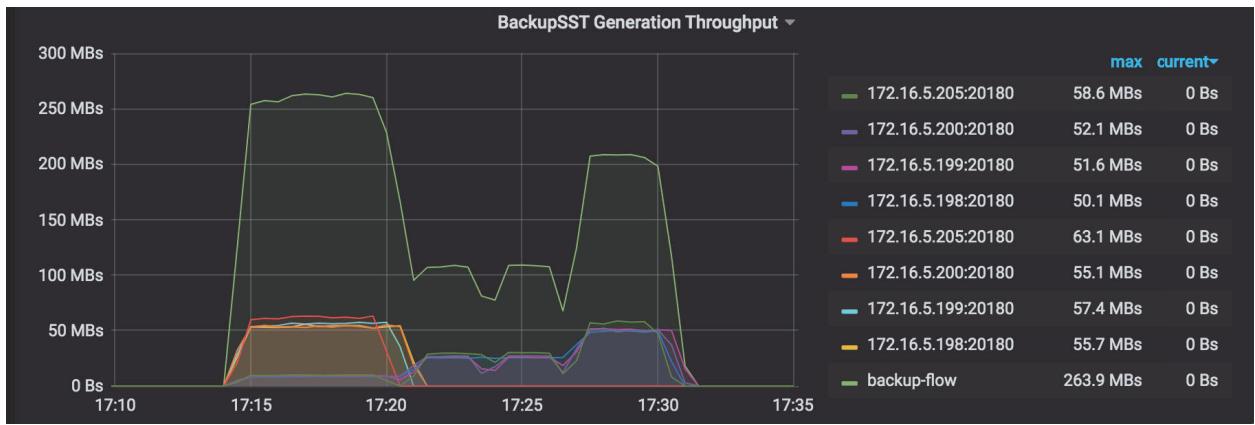


Figure 124: img

**One Backup Range Duration:** the duration of backing up a range, which is the total time cost of scanning KVs and storing the range as the backupSST file.

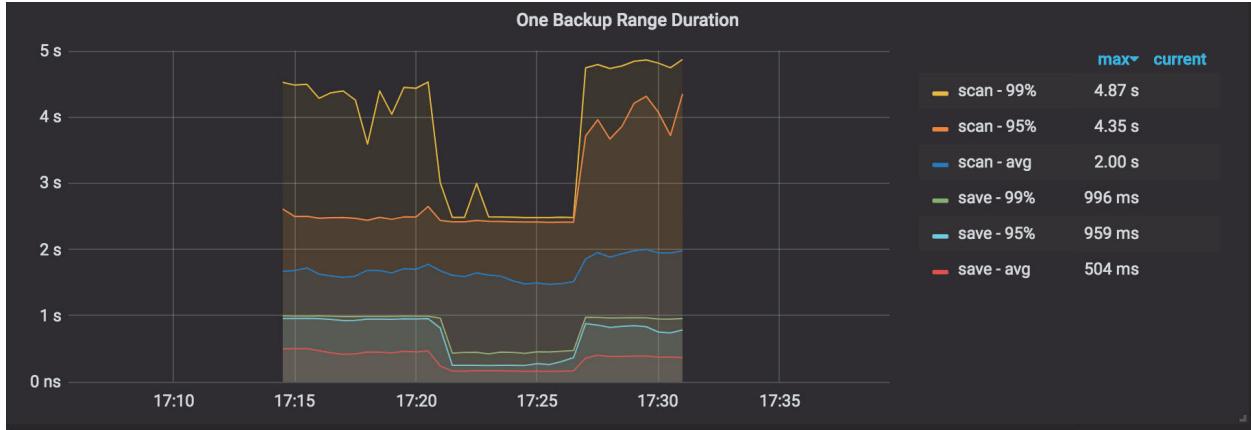


Figure 125: img

**One Backup Subtask Duration:** the duration of each sub-task into which a backup task is divided.

**Note:**

- In this task, the single table to be backed up has three indexes and the task is normally divided into four sub-tasks.
- The panel in the following image has thirteen points on it, which means nine (namely, 13-4) retries. Region scheduling might occur during the backup process, so a few retries is normal.

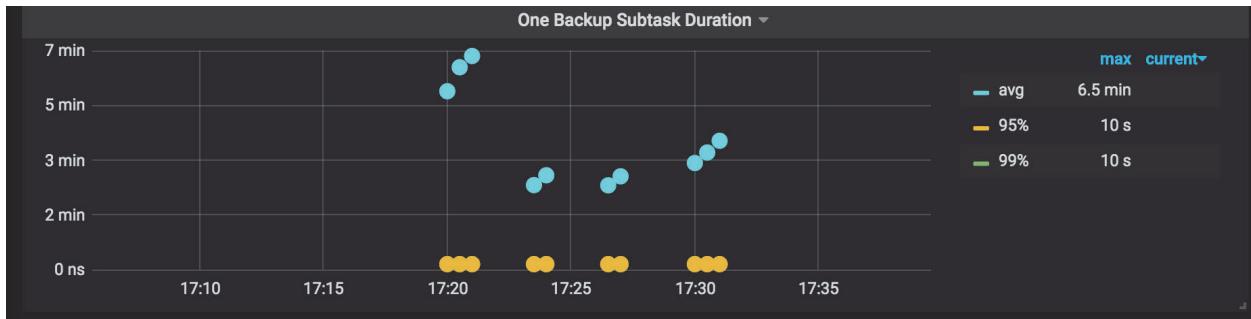


Figure 126: img

**Backup Errors:** the errors occurred during the backup process. No error occurs in normal situations. Even if a few errors occur, the backup operation has the retry mechanism which might increase the backup time but does not affect the operation correctness.

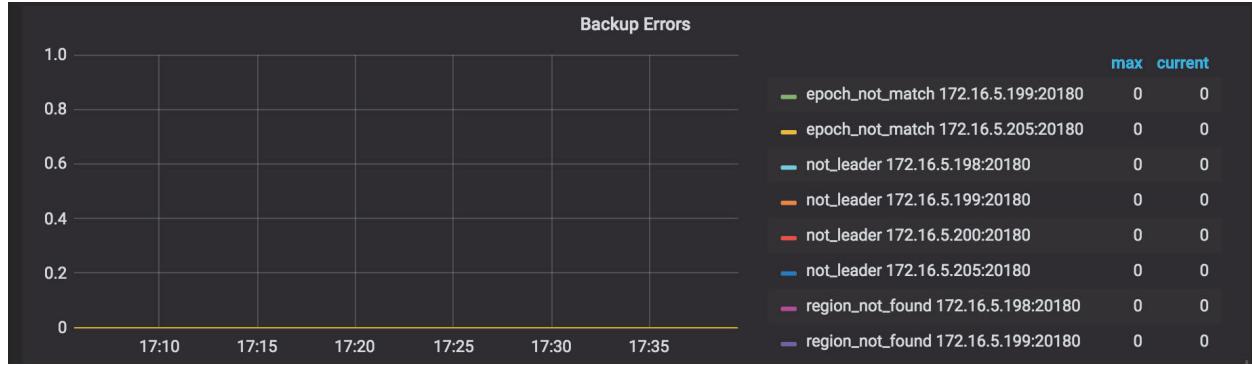


Figure 127: img

**Checksum Request Duration:** the duration of the admin checksum request in the backup cluster.

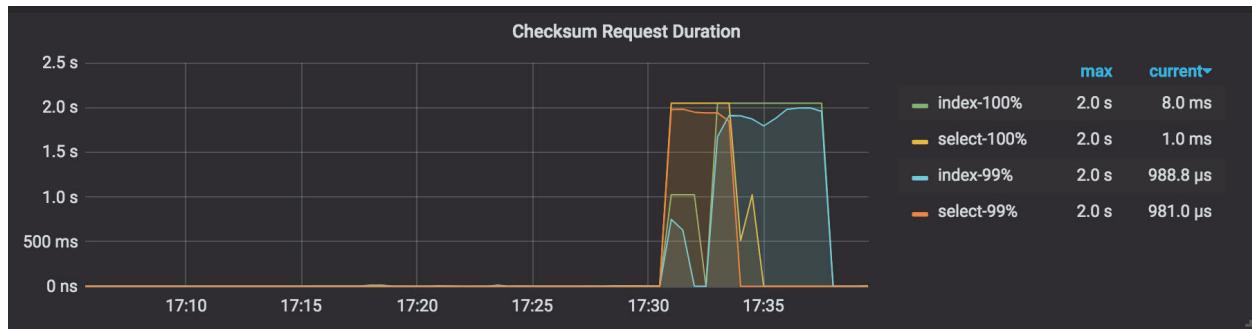


Figure 128: img

### Backup results explanation

When finishing the backup, BR outputs the backup summary to the console.

Before executing the backup command, a path in which the log is stored has been specified. You can get the statistical information of the backup operation from this log. Search “summary” in this log, you can see the following information:

```
["Full backup Success summary:
  total backup ranges: 2,
  total success: 2,
  total failed: 0,
  total take(Full backup time): 31.802912166s,
  total take(real time): 49.799662427s,
  total size(MB): 5997.49,
  avg speed(MB/s): 188.58,
  total kv: 1200000000"]
```

```
["backup checksum"=17.907153678s]
["backup fast checksum"=349.333µs]
["backup total regions"=43]
[BackupTS=422618409346269185]
[Size=826765915]
```

The above log includes the following information:

- Backup duration: `total take(Full backup time)`: 31.802912166s
- Total runtime of the application: `total take(real time)`: 49.799662427s
- Backup data size: `total size(MB)`: 5997.49
- Backup throughput: `avg speed(MB/s)`: 188.58
- Number of backed-up KV pairs: `total kv`: 120000000
- Backup checksum duration: `["backup checksum"=17.907153678s]`
- Total duration of calculating the checksum, KV pairs, and bytes of each table: `["→ backup fast checksum"=349.333µs]`
- Total number of backup Regions: `["backup total regions"=43]`
- The actual size of the backup data in the disk after compression: `[Size=826765915]`
- Snapshot timestamp of the backup data: `[BackupTS=422618409346269185]`

From the above information, the throughput of a single TiKV instance can be calculated:  
 $\text{avg speed(MB/s)}/\text{tikv\_count} = 62.86$ .

#### Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the backup process (for example, in the [Monitoring metrics for the backup](#), the highest CPU usage rate of backup-worker is around 1500% and the overall I/O usage rate is below 30%), you can try to increase the value of `--concurrency` (4 by default) to tune the performance. But this performance tuning method is not suitable for the use cases of many small tables. See the following example:

```
bin/br backup table \
  --db batchmark \
  --table order_line \
  -s local:///br_data/ \
  --pd ${PD_ADDR}:2379 \
  --log-file backup-nfs.log \
  --concurrency 16
```

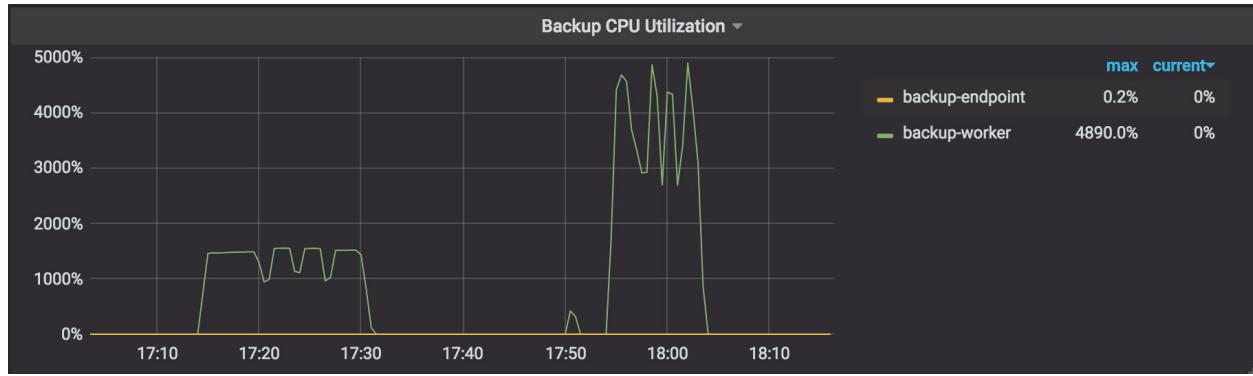


Figure 129: img

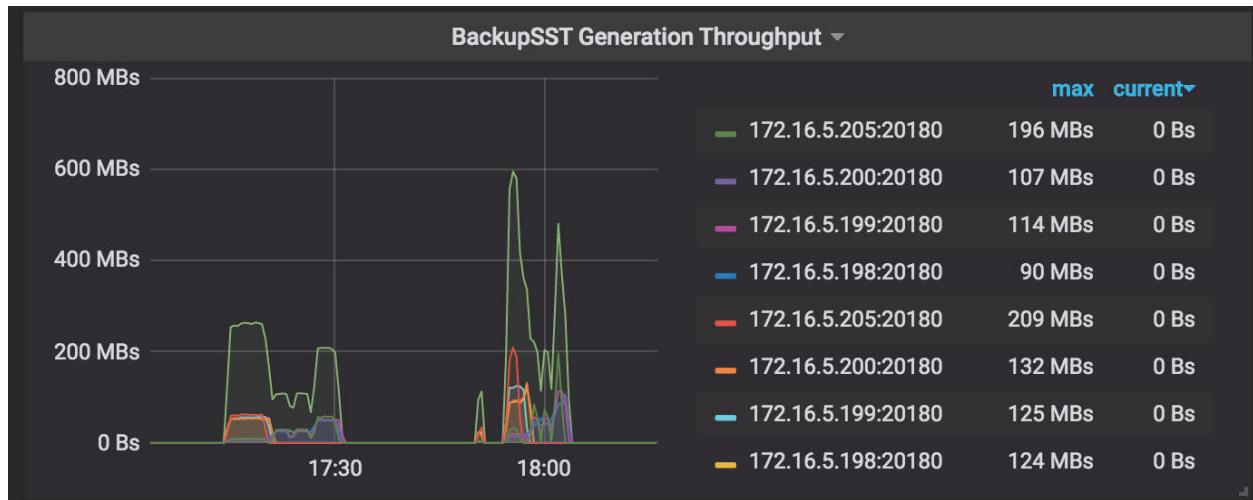


Figure 130: img

The tuned performance results are as follows (with the same data size):

- Backup duration: `total take(s)` reduced from 986.43 to 535.53
- Backup throughput: `avg speed(MB/s)` increased from 358.09 to 659.59
- Throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` increased from 89 to 164.89

#### 10.6.3.3.4 Restore data from a network disk (recommended in production environment)

Use the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

Restoration prerequisites

- Preparation for restoration

Topology

The following diagram shows the typology of BR:

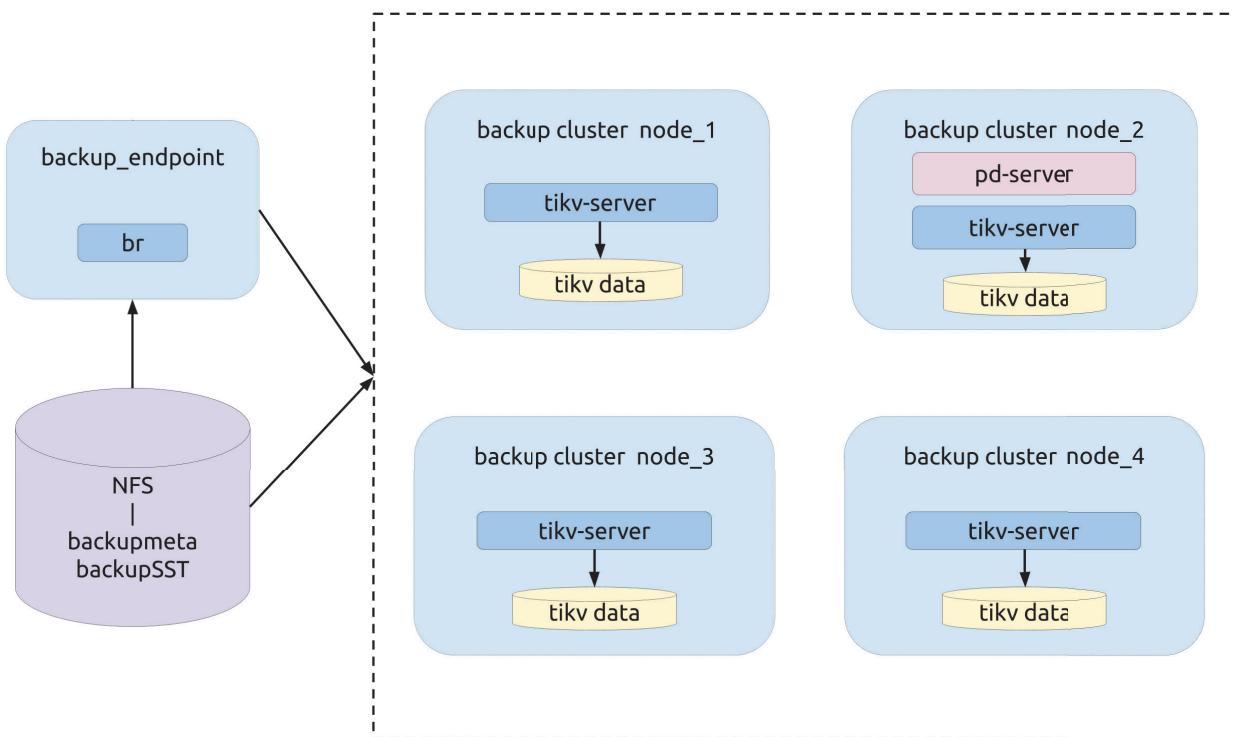


Figure 131: img

Restoration operation

Before the restoration, refer to [Preparation for restoration](#) for the preparation.

Execute the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data
↪ --pd 172.16.5.198:2379 --log-file restore-nfs.log
```

Monitoring metrics for the restoration

During the restoration process, pay attention to the following metrics on the monitoring panels to get the status of the restoration process.

**CPU Utilization:** the CPU usage rate of each working TiKV node in the restoration operation.

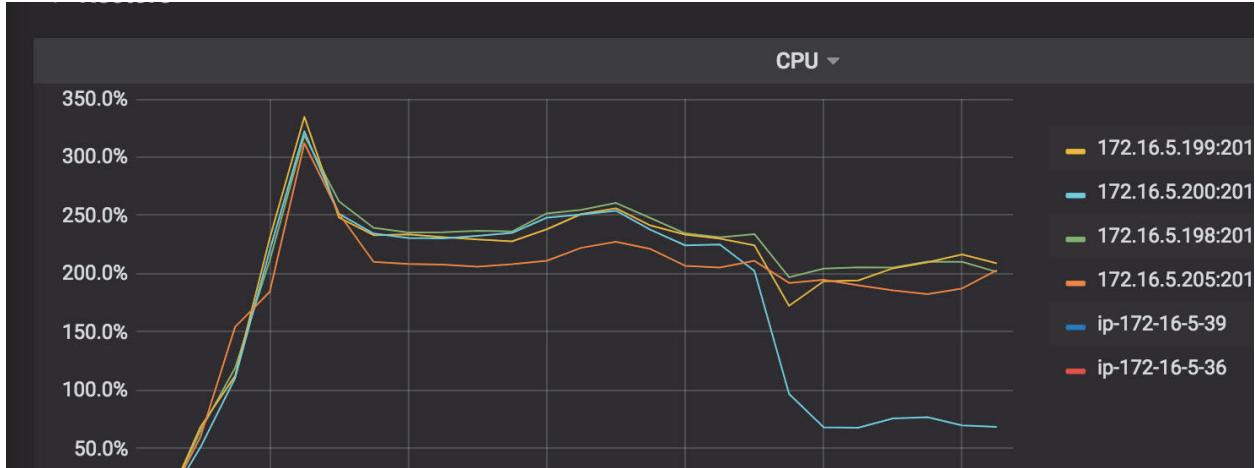


Figure 132: img

**IO Utilization:** the I/O usage rate of each working TiKV node in the restoration operation.

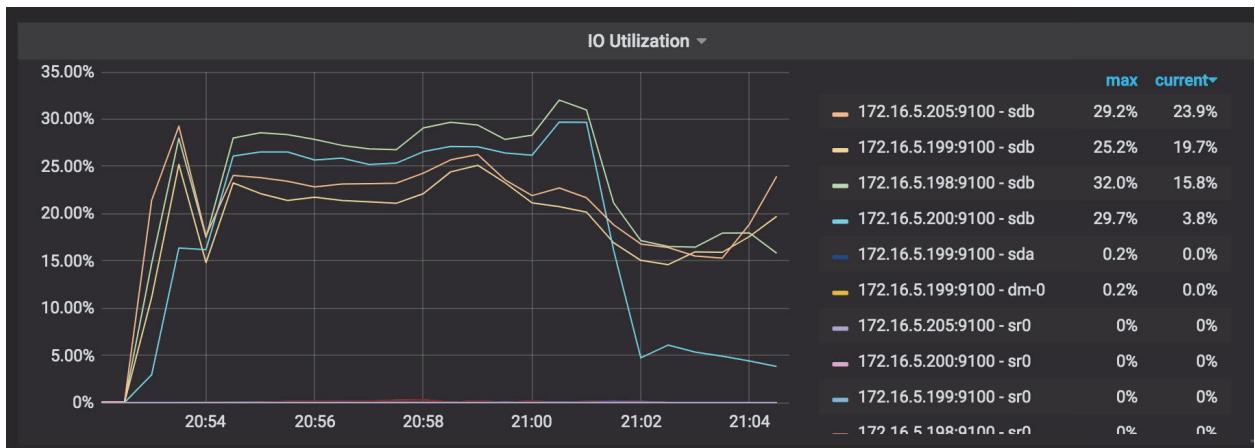


Figure 133: img

**Region:** the Region distribution. The more even Regions are distributed, the better the restoration resources are used.

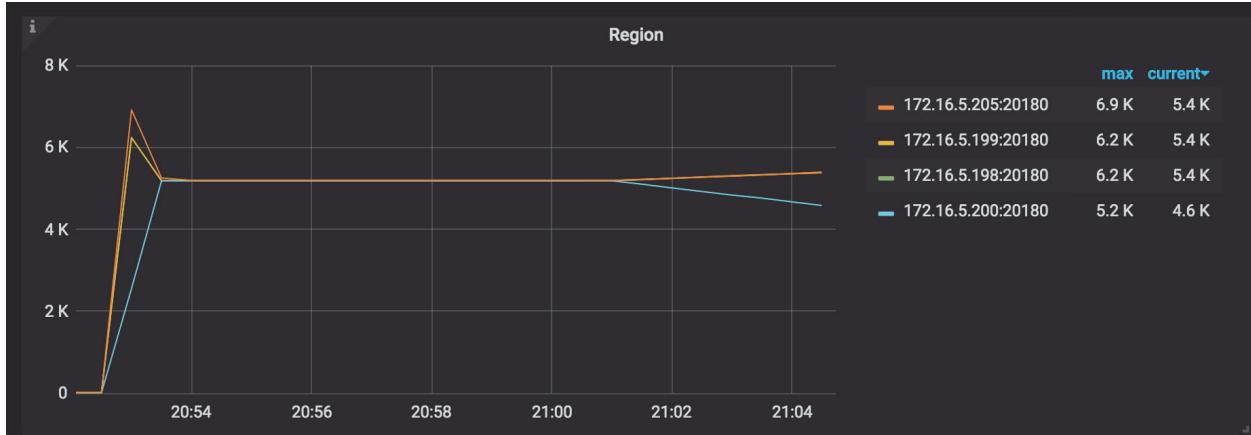


Figure 134: img

**Process SST Duration:** the delay of processing the SST files. When restoring a table, if `tableID` is changed, you need to rewrite `tableID`. Otherwise, `tableID` is renamed. Generally, the delay of rewriting is longer than that of renaming.



Figure 135: img

**DownLoad SST Throughput:** the throughput of downloading SST files from External Storage.

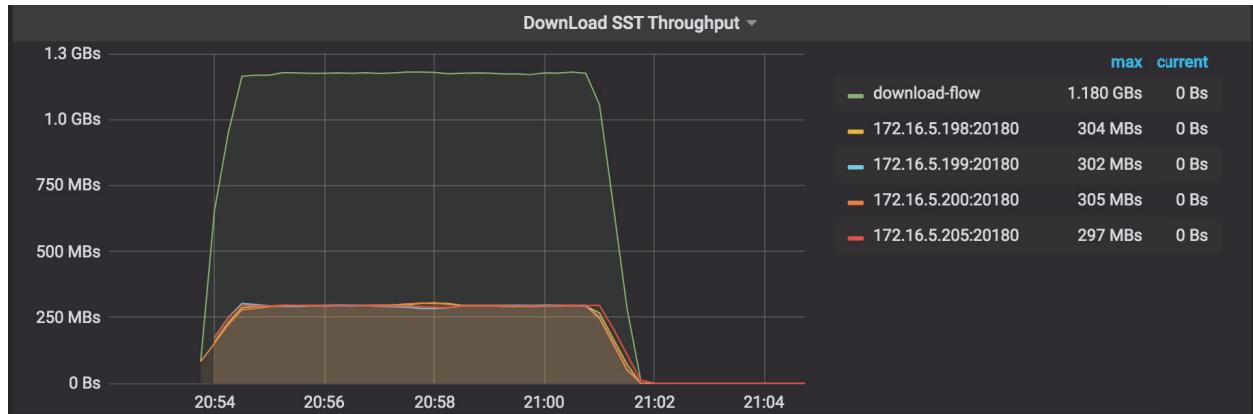


Figure 136: img

**Restore Errors:** the errors occurred during the restoration process.



Figure 137: img

**Checksum Request duration:** the duration of the admin checksum request. This duration for the restoration is longer than that for the backup.

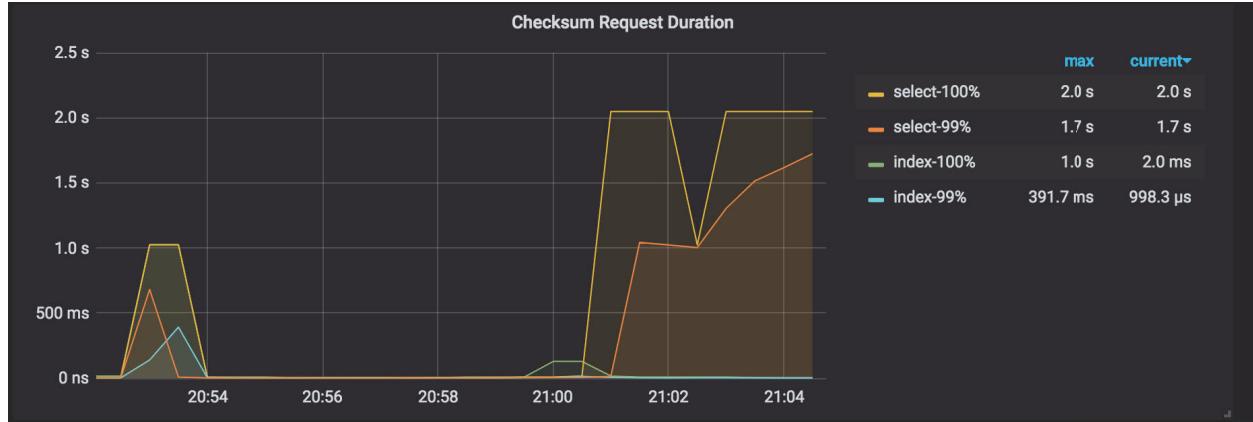


Figure 138: img

### Restoration results explanation

Before executing the restoration command, a path in which the log is stored has been specified. You can get the statistical information of the restoration operation from this log. Search “summary” in this log, you can see the following information:

```
["Table Restore summary:
total restore tables: 1,
total success: 1,
total failed: 0,
total take(Full restore time): 17m1.001611365s,
total take(real time): 16m1.371611365s,
total kv: 5659888624,
total size(MB): 353227.18,
avg speed(MB/s): 367.42"]
[{"restore files":9263}
[{"restore ranges":6888}
[{"split region":49.049182743s}
[{"restore checksum":6m34.879439498s}
[Size=48693068713]
```

The above log includes the following information:

- Restore duration: `total take(Full restore time): 17m1.001611365s`
- Total runtime of the application: `total take(real time): 16m1.371611365s`
- Restore data size: `total size(MB): 353227.18`
- Restore KV pair number: `total kv: 5659888624`
- Restore throughput: `avg speed(MB/s): 367.42`
- Region Split duration: `take=49.049182743s`
- Restore checksum duration: `restore checksum=6m34.879439498s`

- The actual size of the restored data in the disk: [Size=48693068713]

From the above information, the following items can be calculated:

- The throughput of a single TiKV instance:  $\text{avg speed(MB/s)}/\text{tikv\_count} = 91.8$
- The average restore speed of a single TiKV instance:  $\text{total size(MB)}/(\text{split time} + \text{restore time})/\text{tikv\_count} = 87.4$

### Performance tuning

If the resource usage of TiKV does not become an obvious bottleneck during the restore process, you can try to increase the value of `--concurrency` which is 128 by default. See the following example:

```
bin/br restore table --db batchmark --table order_line -s local:///br_data/
    ↪ --pd 172.16.5.198:2379 --log-file restore-concurrency.log --
    ↪ concurrency 1024
```

The tuned performance results are as follows (with the same data size):

- Restore duration: `total take(s)` reduced from 961.37 to 443.49
- Restore throughput: `avg speed(MB/s)` increased from 367.42 to 796.47
- Throughput of a single TiKV instance: `avg speed(MB/s)/tikv_count` increased from 91.8 to 199.1
- Average restore speed of a single TiKV instance: `total size(MB)/(split time + restore time)/tikv_count` increased from 87.4 to 162.3

#### 10.6.3.3.5 Back up a single table to a local disk (recommended in testing environment)

Use the `br backup` command to back up the single table `--db batchmark --table order_line` to the specified path `local:///home/tidb/backup_local` in the local disk.

##### Backup prerequisites

- Preparation for backup
- Each TiKV node has a separate disk to store the `backupSST` file.
- The `backup_endpoint` node has a separate disk to store the `backupmeta` file.
- TiKV and the `backup_endpoint` node must have the same directory for the backup (for example, `/home/tidb/backup_local`).

##### Topology

The following diagram shows the typology of BR:

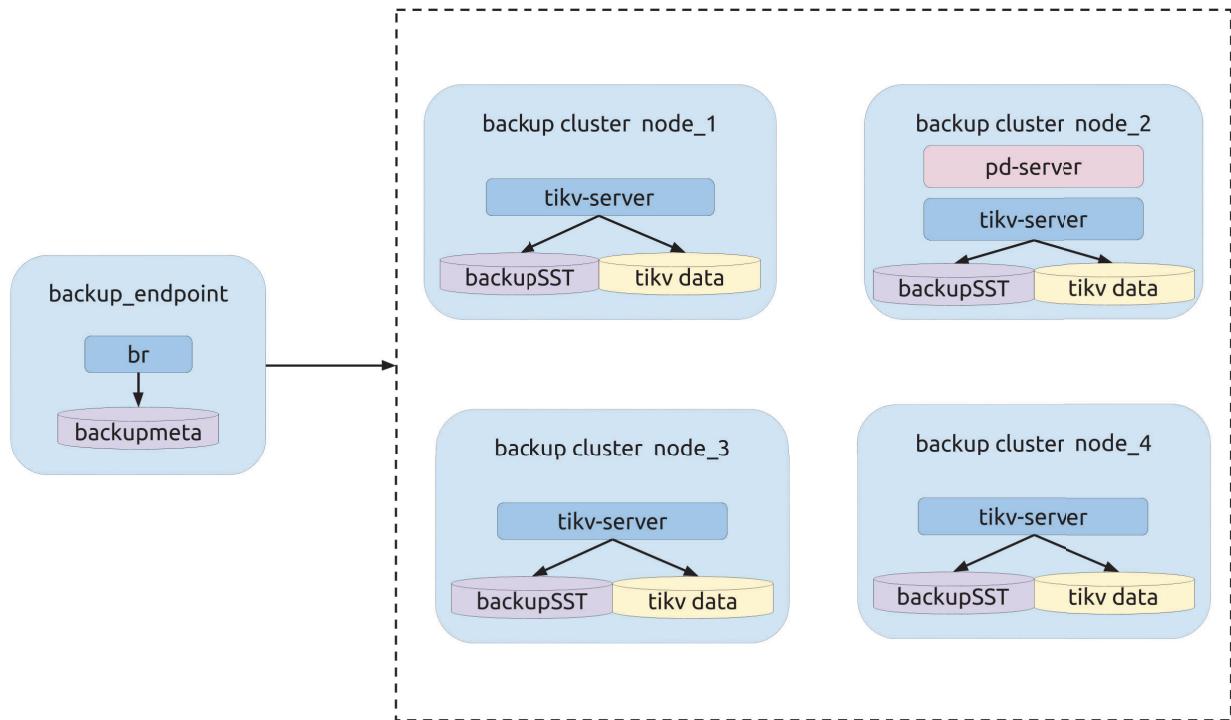


Figure 139: img

### Backup operation

Before the backup operation, execute the `admin checksum table order_line` command to get the statistical information of the table to be backed up (`--db batchmark`  $\rightarrow$  `--table order_line`). The following image shows an example of this information:

Db_name	Table_name	Checksum_crc64_xor	Total_kvs	Total_bytes
batchmark	order_line	10912722838344822475	5659888624	370385538778

1 row in set (5 min 47.59 sec)

Figure 140: img

Execute the `br backup` command:

```
bin/br backup table \
--db batchmark \
--table order_line \
-s local:///home/tidb/backup_local/ \
```

```
--pd ${PD_ADDR}:2379 \
--log-file backup_local.log
```

During the backup process, pay attention to the metrics on the monitoring panels to get the status of the backup process. See [Monitoring metrics for the backup](#) for details.

#### Backup results explanation

Before executing the backup command, a path in which the log is stored has been specified. You can get the statistical information of the backup operation from this log. Search “summary” in this log, you can see the following information:

```
["Table backup summary: total backup ranges: 4, total success: 4, total
 ↪ failed: 0, total take(s): 551.31, total kv: 5659888624, total size(MB
 ↪ ): 353227.18, avg speed(MB/s): 640.71"] ["backup total regions"=6795]
 ↪ ["backup checksum"=6m33.962719217s] ["backup fast checksum
 ↪ "=22.995552ms]
```

The information from the above log includes:

- Backup duration: `total take(s): 551.31`
- Data size: `total size(MB): 353227.18`
- Backup throughput: `avg speed(MB/s): 640.71`
- Backup checksum duration: `take=6m33.962719217s`

From the above information, the throughput of a single TiKV instance can be calculated:  $\text{avg speed(MB/s)}/\text{tikv\_count} = 160$ .

#### 10.6.3.3.6 Restore data from a local disk (recommended in testing environment)

Use the `br restore` command to restore the complete backup data to an offline cluster. Currently, BR does not support restoring data to an online cluster.

##### Restoration prerequisites

- [Preparation for restoration](#)
- The TiKV cluster and the backup data do not have a duplicate database or table. Currently, BR does not support table route.
- Each TiKV node has a separate disk to store the `backupSST` file.
- The `restore_endpoint` node has a separate disk to store the `backupmeta` file.
- TiKV and the `restore_endpoint` node must have the same directory for the restoration (for example, `/home/tidb/backup_local/`).

Before the restoration, follow these steps:

1. Collect all backupSST files into the same directory.
2. Copy the collected backupSST files to all TiKV nodes of the cluster.
3. Copy the `backupmeta` file to the `restore endpoint` node.

## Topology

The following diagram shows the typology of BR:

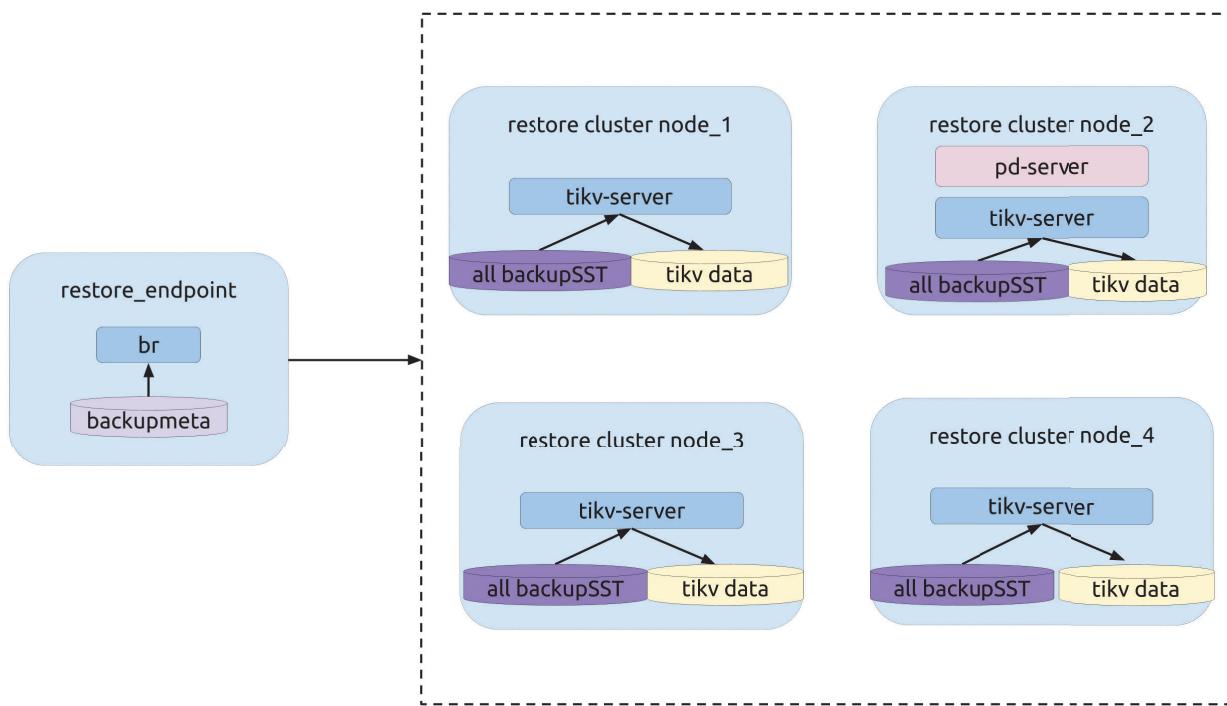


Figure 141: img

## Restoration operation

Execute the `br restore` command:

```
bin/br restore table --db batchmark --table order_line -s local:///home/tidb
↪ /backup_local/ --pd 172.16.5.198:2379 --log-file restore_local.log
```

During the restoration process, pay attention to the metrics on the monitoring panels to get the status of the restoration process. See [Monitoring metrics for the restoration](#) for details.

## Restoration results explanation

Before executing the restoration command, a path in which the log is stored has been specified. You can get the statistical information of the restoration operation from this log. Search “summary” in this log, you can see the following information:

```
[{"Table Restore summary: total restore tables: 1, total success: 1, total
 ↪ failed: 0, total take(s): 908.42, total kv: 5659888624, total size(MB
 ↪ ): 353227.18, avg speed(MB/s): 388.84"] [{"restore files":9263} [{"restore ranges":6888} [{"split region":58.7885518s} [{"restore checksum
 ↪ "":6m19.349067937s}]]]
```

The above log includes the following information:

- Restoration duration: total take(s): 908.42
- Data size: total size(MB): 353227.18
- Restoration throughput: avg speed(MB/s): 388.84
- Region Split duration: take=58.7885518s
- Restoration checksum duration: take=6m19.349067937s

From the above information, the following items can be calculated:

- The throughput of a single TiKV instance: avg speed(MB/s)/tikv\_count = 97.2
- The average restoration speed of a single TiKV instance: total size(MB)/(split
 ↪ time + restore time)/tikv\_count = 92.4

#### 10.6.3.4 Error handling during backup

This section introduces the common errors occurred during the backup process.

##### 10.6.3.4.1 key locked Error in the backup log

Error message in the log: log - ["backup occur kv error"] [error="{"KvError
 ↪ ":"locked"}":

If a key is locked during the backup process, BR tries to resolve the lock. A small number of these errors do not affect the correctness of the backup.

##### 10.6.3.4.2 Backup failure

Error message in the log: log - Error: msg:"Io(Custom { kind: AlreadyExists,
 ↪ error: "[5\_5359\_42\_123\_default.sst] is already exists in /dir/backup\_local
 ↪ /" })"

If the backup operation fails and the above message occurs, perform one of the following operations and then start the backup operation again:

- Change the directory for the backup. For example, change /dir/backup-2020-01-01/ to /dir/backup\_local/.
- Delete the backup directory of all TiKV nodes and BR nodes.

#### 10.6.4 External Storages

Backup & Restore (BR), TiDB Lighting, and Dumpling support reading and writing data on the local filesystem and on Amazon S3. BR also supports reading and writing data on the Google Cloud Storage (GCS). These are distinguished by the URL scheme in the `--storage` parameter passed into BR, in the `-d` parameter passed into TiDB Lightning, and in the `--output` (`-o`) parameter passed into Dumpling.

##### 10.6.4.1 Schemes

The following services are supported:

Service	Schemes	Example URL
Local filesystem, distributed on every node	local	local:///path/to/dest/
Amazon S3 and compatible services	s3	s3://bucket-name/prefix/of/dest/
Google Cloud Storage (GCS)	gcs, gs	gcs://bucket-name/prefix/of/dest/
Write to nowhere (for benchmarking only)	noop	noop://

##### 10.6.4.2 URL parameters

Cloud storages such as S3 and GCS sometimes require additional configuration for connection. You can specify parameters for such configuration. For example:

- Use Dumpling to export data to S3:

```
./dumpling -u root -h 127.0.0.1 -P 3306 -B mydb -F 256MiB \
-o 's3://my-bucket/sql-backup?region=us-west-2'
```

- Use TiDB Lightning to import data from S3:

```
./tidb-lightning --tidb-port=4000 --pd-urls=127.0.0.1:2379 --backend=
↪ local --sorted-kv-dir=/tmp/sorted-kvs \
-d 's3://my-bucket/sql-backup?region=us-west-2'
```

- Use TiDB Lightning to import data from S3 (using the path style in the request mode):

```
./tidb-lightning --tidb-port=4000 --pd-urls=127.0.0.1:2379 --backend=
↪ local --sorted-kv-dir=/tmp/sorted-kvs \
-d 's3://my-bucket/sql-backup?force-path-style=true&endpoint=http
↪ ://10.154.10.132:8088'
```

- Use BR to back up data to GCS:

```
./br backup full -u 127.0.0.1:2379 \
-s 'gcs://bucket-name/prefix'
```

#### 10.6.4.2.1 S3 URL parameters

URL parameter	Description
<code>access-key</code>	The access key
<code>secret-</code>	The secret
<code>→ access-</code>	access key
<code>→ key</code>	
<code>region</code>	Service Region for Amazon S3 (default to <code>us-east-1</code> )
<code>→ )</code>	
<code>use-</code>	Whether
<code>→ accelerate</code>	to use the accelerate
<code>→ -</code>	
<code>→ endpoint</code>	endpoint
<code>→ on</code>	
<code>endpoint</code>	Amazon S3 (default to <code>false</code> )
<code>endpoint</code>	URL of custom endpoint for S3-compatible services (for example, <code>https://s3.example.com/</code> )

URL parameter	Description
<b>force-path</b> ↪ <b>-style</b>	Use path style access rather than virtual hosted style access (default to <b>false</b> )
<b>storage-</b> ↪ <b>class</b>	Storage class of the uploaded objects (for example, <b>STANDARD</b> , <b>STANDARD_IA</b> ↪ )
<b>sse</b>	Server-side encryption algorithm used to encrypt the upload (empty, <b>AES256</b> or <b>aws:kms</b> )
<b>sse-kms-</b> ↪ <b>key-id</b>	If <b>sse</b> is set to <b>aws:kms</b> , specifies the KMS ID

URL parameter	Description
acl	Canned ACL of the uploaded objects (for example, private, authenticated ↪ -read)

**Note:**

It is not recommended to pass in the access key and secret access key directly in the storage URL, because these keys are logged in plain text. The migration tools try to infer these keys from the environment in the following order:

1. \$AWS\_ACCESS\_KEY\_ID and \$AWS\_SECRET\_ACCESS\_KEY environment variables
2. \$AWS\_ACCESS\_KEY and \$AWS\_SECRET\_KEY environment variables
3. Shared credentials file on the tool node at the path specified by the \$AWS\_SHARED\_CREDENTIALS\_FILE  
↪ environment variable
4. Shared credentials file on the tool node at `~/.aws/credentials`
5. Current IAM role of the Amazon EC2 container
6. Current IAM role of the Amazon ECS task

#### 10.6.4.2.2 GCS URL parameters

URL parameter	Description
credentials ↪ -file	The path to the credentials JSON file on the tool node

URL parameter	Description
<code>storage-class</code>	Storage class of the uploaded objects (for example, STANDARD, COLDLINE)
<code>predefined-acl</code>	Predefined ACL of the uploaded objects (for example, private, project- ↳ private ↳ )

When `credentials-file` is not specified, the migration tool will try to infer the credentials from the environment, in the following order:

1. Content of the file on the tool node at the path specified by the `$GOOGLE_APPLICATION_CREDENTIALS` environment variable
2. Content of the file on the tool node at `~/.config/gcloud/application_default_credentials.json`
3. When running in GCE or GAE, the credentials fetched from the metadata server.

#### 10.6.4.3 Command-line parameters

In addition to the URL parameters, BR and Dumpling also support specifying these configurations using command-line parameters. For example:

```
./dumpling -u root -h 127.0.0.1 -P 3306 -B mydb -F 256MiB \
-o 's3://my-bucket/sql-backup' \
--s3.region 'us-west-2'
```

If you have specified URL parameters and command-line parameters at the same time, the URL parameters are overwritten by the command-line parameters.

##### 10.6.4.3.1 S3 command-line parameters

---

Command-line parameter	Description
--s3.	Amazon S3's service region, which de-faults to us-
↳ <b>region</b>	→ east → -1.
--s3.	The URL of custom endpoint for S3-compatible services.
↳ <b>endpoint</b>	For example, https → :// → s3. → example → . → com → /.

---

Command-line parameter	Description
--s3. ↳ <b>storage</b> ↳ <b>-class</b>	The storage class of the upload object. For example, STANDARD ↳ and STANDARD_IA ↳ .
--s3.sse	The server-side encryption algorithm used to encrypt the upload. The value options are empty, AES256 and aws: ↳ kms ↳ .

---

Command-line parameter	Description
--s3.sse- ↳ kms-key ↳ -id	If --s3. is con- figured as aws: ↳ kms ↳ , this parameter is used to specify the KMS ID.
--s3.acl	The canned ACL of the upload object. For ex- ample, <b>private</b> ↳ and <b>authenticated</b> ↳ - ↳ <b>read</b> ↳ .

---

Command-line parameter	Description
--s3. → provider	The type of the S3-compatible service. The supported types are aws, alibaba (, ceph, netease) and other.

---

#### 10.6.4.3.2 GCS command-line parameters

---

Command-line parameter	Description
--gcs. → credentials → -file	The path of the JSON-formatted credential on the tool node.

---

Command-line parameter	Description
--gcs . ↪ storage ↪ -class	The storage type of the upload object, such as STANDARD and COLDLINE.
--gcs . ↪ predefined ↪ -acl	The predefined ACL of the upload object, such as private and project- ↪ private ↪ .

#### 10.6.4.4 BR sending credentials to TiKV

By default, when using S3 and GCS destinations, BR will send the credentials to every TiKV nodes to reduce setup complexity.

However, this is unsuitable on cloud environment, where every node has their own role and permission. In such cases, you need to disable credentials sending with --send-credentials-to-tikv=false (or the short form -c=0):

```
./br backup full -c=0 -u pd-service:2379 -s 's3://bucket-name/prefix'
```

When using SQL statements to `back up` and `restore` data, you can add the `SEND_CREDENTIALS_TO_TIKV = FALSE` option:

```
BACKUP DATABASE * TO 's3://bucket-name/prefix' SEND_CREDENTIALS_TO_TIKV =
↪ FALSE;
```

This option is not supported in TiDB Lightning and Dumpling, because the two applications are currently standalone.

## 10.6.5 Backup & Restore FAQ

This document lists the frequently asked questions (FAQs) and the solutions about Backup & Restore (BR).

### 10.6.5.1 What should I do if the error message could not read local://...:download sst failed is returned during data restoration?

When you restore data, each node must have access to **all** backup files (SST files). By default, if local storage is used, you cannot restore data because the backup files are scattered among different nodes. Therefore, you have to copy the backup file of each TiKV node to the other TiKV nodes.

It is recommended to mount an NFS disk as a backup disk during backup. For details, see [Back up a single table to a network disk](#).

### 10.6.5.2 How much does it affect the cluster during backup using BR?

When you use the `oltp_read_only` scenario of `sysbench` to back up to a disk (make sure the backup disk and the service disk are different) at full rate, the cluster QPS is decreased by 15%-25%. The impact on the cluster depends on the table schema.

To reduce the impact on the cluster, you can use the `--ratelimit` parameter to limit the backup rate.

### 10.6.5.3 Does BR back up system tables? During data restoration, do they raise conflict?

Before v5.1.0, BR filtered out data from the system schema `mysql` during the backup. Since v5.1.0, BR **backs up** all data by default, including the system schemas `mysql.*`. But the technical implementation of restoring the system tables in `mysql.*` is not complete yet, so the tables in the system schema `mysql` are **not** restored by default. For more details, refer to the [Back up and restore table data in the mysql system schema \(experimental feature\)](#).

### 10.6.5.4 What should I do to handle the Permission denied or No such file or directory error, even if I have tried to run BR using root in vain?

You need to confirm whether TiKV has access to the backup directory. To back up data, confirm whether TiKV has the write permission. To restore data, confirm whether it has the read permission.

During the backup operation, if the storage medium is the local disk or a network file system (NFS), make sure that the user to start BR and the user to start TiKV are consistent (if BR and TiKV are on different machines, the users' UIDs must be consistent). Otherwise, the `Permission denied` issue might occur.

Running BR with the root access might fail due to the disk permission, because the backup files (SST files) are saved by TiKV.

**Note:**

You might encounter the same problem during data restoration. When the SST files are read for the first time, the read permission is verified. The execution duration of DDL suggests that there might be a long interval between checking the permission and running BR. You might receive the error message `Permission denied` after waiting for a long time.

Therefore, it is recommended to check the permission before data restore according to the following steps:

1. Run the Linux-native command for process query:

```
ps aux | grep tikv-server
```

The output of the above command:

```
tidb_ouo 9235 10.9 3.8 2019248 622776 ? Ssl 08:28 1:12 bin/tikv-
    ↳ server --addr 0.0.0.0:20162 --advertise-addr 172.16.6.118:20162
    ↳ --status-addr 0.0.0.0:20188 --advertise-status-addr
    ↳ 172.16.6.118:20188 --pd 172.16.6.118:2379 --data-dir /home/user1/
    ↳ tidb-data/tikv-20162 --config conf/tikv.toml --log-file /home/
    ↳ user1/tidb-deploy/tikv-20162/log/tikv.log
tidb_ouo 9236 9.8 3.8 2048940 631136 ? Ssl 08:28 1:05 bin/tikv-
    ↳ server --addr 0.0.0.0:20161 --advertise-addr 172.16.6.118:20161
    ↳ --status-addr 0.0.0.0:20189 --advertise-status-addr
    ↳ 172.16.6.118:20189 --pd 172.16.6.118:2379 --data-dir /home/user1/
    ↳ tidb-data/tikv-20161 --config conf/tikv.toml --log-file /home/
    ↳ user1/tidb-deploy/tikv-20161/log/tikv.log
```

Or you can run the following command:

```
ps aux | grep tikv-server | awk '{print $1}'
```

The output of the above command:

```
tidb_ouo
tidb_ouo
```

2. Query the startup information of the cluster using the TiUP command:

```
tiup cluster list
```

The output of the above command:

```
[root@Copy-of-VM-EE-CentOS76-v1 br]# tiup cluster list
Starting component `cluster`: /root/.tiup/components/cluster/v1.5.2/
  ↳ tiup-cluster list
Name      User      Version Path
  ↳
-----  -----  -----
  ↳
tidb_cluster tidb_ouo v5.0.2 /root/.tiup/storage/cluster/clusters/
  ↳ tidb_cluster /root/.tiup/storage/cluster/clusters/tidb_cluster/
  ↳ ssh/id_rsa
```

3. Check the permission for the backup directory. For example, `backup` is for backup data storage:

```
ls -al backup
```

The output of the above command:

```
[root@Copy-of-VM-EE-CentOS76-v1 user1]# ls -al backup
total 0
drwxr-xr-x 2 root root 6 Jun 28 17:48 .
drwxr-xr-x 11 root root 310 Jul 4 10:35 ..
```

From the above output, you can find that the `tikv-server` instance is started by the user `tidb_ouo`. But the user `tidb_ouo` does not have the write permission for `backup`, the backup fails.

#### 10.6.5.5 What should I do to handle the `Io(Os...)` error?

Almost all of these problems are system call errors that occur when TiKV writes data to the disk. For example, if you encounter error messages such as `Io(Os {code: 13, kind: : PermissionDenied...})` or `Io(Os {code: 2, kind: NotFound...})`, you can first check the mounting method and the file system of the backup directory, and try to back up data to another folder or another hard disk.

For example, you might encounter the `Code: 22(invalid argument)` error when backing up data to the network disk built by `samba`.

#### 10.6.5.6 What should I do to handle the `rpc error: code = Unavailable desc =... error occurred in BR?`

This error might occur when the capacity of the cluster to restore (using BR) is insufficient. You can further confirm the cause by checking the monitoring metrics of this cluster or the TiKV log.

To handle this issue, you can try to scale out the cluster resources, reduce the concurrency during restore, and enable the `RATE_LIMIT` option.

#### 10.6.5.7 Where are the backed up files stored when I use local storage?

When you use `local` storage, `backupmeta` is generated on the node where BR is running, and backup files are generated on the Leader nodes of each Region.

#### 10.6.5.8 How about the size of the backup data? Are there replicas of the backup?

During data backup, backup files are generated on the Leader nodes of each Region. The size of the backup is equal to the data size, with no redundant replicas. Therefore, the total data size is approximately the total number of TiKV data divided by the number of replicas.

However, if you want to restore data from local storage, the number of replicas is equal to that of the TiKV nodes, because each TiKV must have access to all backup files.

#### 10.6.5.9 What should I do when BR restores data to the upstream cluster of TiCDC/Drainer?

- **The data restored using BR cannot be replicated to the downstream.** This is because BR directly imports SST files but the downstream cluster currently cannot obtain these files from the upstream.
- Before v4.0.3, DDL jobs generated during the BR restore might cause unexpected DDL executions in TiCDC/Drainer. Therefore, if you need to perform restore on the upstream cluster of TiCDC/Drainer, add all tables restored using BR to the TiCDC/Drainer block list.

You can use `filter.rules` to configure the block list for TiCDC and use `syncer.ignore ↩ -table` to configure the block list for Drainer.

#### 10.6.5.10 Does BR back up the `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` information of a table? Does the restored table have multiple Regions?

Yes. BR backs up the `SHARD_ROW_ID_BITS` and `PRE_SPLIT_REGIONS` information of a table. The data of the restored table is also split into multiple Regions.

#### 10.6.5.11 Why is the `region is unavailable` error reported for a SQL query after I use BR to restore the backup data?

If the cluster backed up using BR has TiFlash, `TableInfo` stores the TiFlash information when BR restores the backup data. If the cluster to be restored does not have TiFlash, the `region is unavailable` error is reported.

#### 10.6.5.12 Does BR support in-place full recovery of some historical backup?

No. BR does not support in-place full recovery of some historical backup.

#### 10.6.5.13 How can I use BR for incremental backup in the Kubernetes environment?

To get the `commitTs` field of the last BR backup, run the `kubectl -n ${namespace} → } get bk ${name}` command using kubectl. You can use the content of this field as `--lastbackups`.

#### 10.6.5.14 How can I convert BR backupTS to Unix time?

BR `backupTS` defaults to the latest timestamp obtained from PD before the backup starts. You can use `pd-ctl tso timestamp` to parse the timestamp to obtain an accurate value, or use `backupTS >> 18` to quickly obtain an estimated value.

#### 10.6.5.15 After BR restores the backup data, do I need to execute the ANALYZE statement on the table to update the statistics of TiDB on the tables and indexes?

BR does not back up statistics (except in v4.0.9). Therefore, after restoring the backup data, you need to manually execute `ANALYZE TABLE` or wait for TiDB to automatically execute `ANALYZE`.

In v4.0.9, BR backs up statistics by default, which consumes too much memory. To ensure that the backup process goes well, the backup for statistics is disabled by default starting from v4.0.10.

If you do not execute `ANALYZE` on the table, TiDB will fail to select the optimized execution plan due to inaccurate statistics. If query performance is not a key concern, you can ignore `ANALYZE`.

#### 10.6.5.16 Can I use multiple BR processes at the same time to restore the data of a single cluster?

**It is strongly not recommended** to use multiple BR processes at the same time to restore the data of a single cluster for the following reasons:

- When BR restores data, it modifies some global configurations of PD. Therefore, if you use multiple BR processes for data restore at the same time, these configurations might be mistakenly overwritten and cause abnormal cluster status.
- BR consumes a lot of cluster resources to restore data, so in fact, running BR processes in parallel improves the restore speed only to a limited extent.
- There has been no test for running multiple BR processes in parallel for data restore, so it is not guaranteed to succeed.

## 10.7 TiDB Binlog

### 10.7.1 TiDB Binlog Cluster Overview

This document introduces the architecture and the deployment of the cluster version of TiDB Binlog.

TiDB Binlog is a tool used to collect binlog data from TiDB and provide near real-time backup and replication to downstream platforms.

TiDB Binlog has the following features:

- **Data replication:** replicate the data in the TiDB cluster to other databases
- **Real-time backup and restoration:** back up the data in the TiDB cluster and restore the TiDB cluster when the cluster fails

**Note:**

TiDB Binlog is not compatible with some features introduced in TiDB v5.0 and they cannot be used together. For details, see [Notes](#). It is recommended to use [TiCDC](#) instead of TiDB Binlog.

#### 10.7.1.1 TiDB Binlog architecture

The TiDB Binlog architecture is as follows:

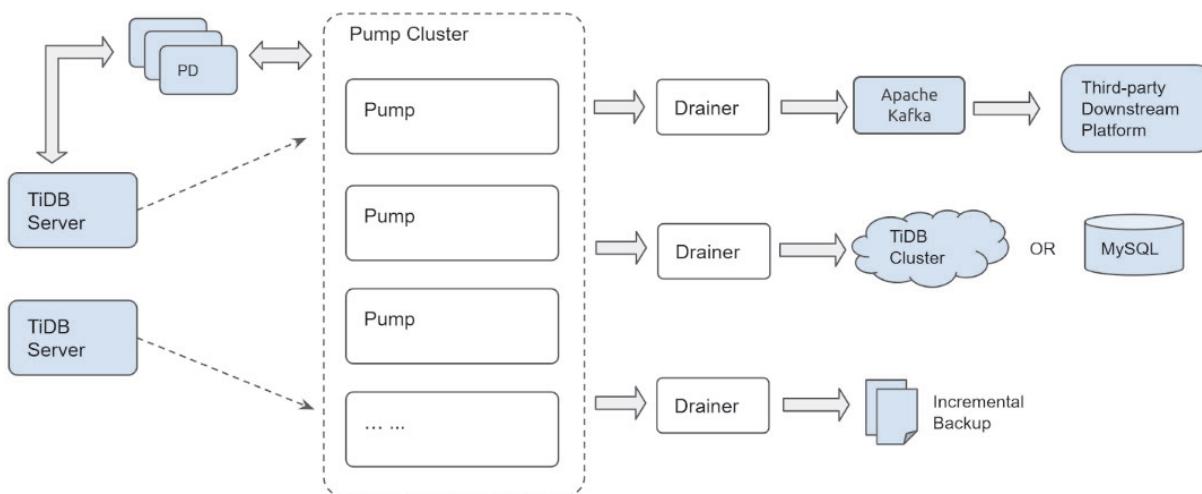


Figure 142: TiDB Binlog architecture

The TiDB Binlog cluster is composed of Pump and Drainer.

#### 10.7.1.1.1 Pump

Pump is used to record the binlogs generated in TiDB, sort the binlogs based on the commit time of the transaction, and send binlogs to Drainer for consumption.

#### 10.7.1.1.2 Drainer

Drainer collects and merges binlogs from each Pump, converts the binlog to SQL or data of a specific format, and replicates the data to a specific downstream platform.

#### 10.7.1.1.3 binlogctl guide

`binlogctl` is an operations tool for TiDB Binlog with the following features:

- Obtaining the current `tso` of TiDB cluster
- Checking the Pump/Drainer state
- Modifying the Pump/Drainer state
- Pausing or closing Pump/Drainer

#### 10.7.1.2 Main features

- Multiple Pumps form a cluster which can scale out horizontally
- TiDB uses the built-in Pump Client to send the binlog to each Pump
- Pump stores binlogs and sends the binlogs to Drainer in order
- Drainer reads binlogs of each Pump, merges and sorts the binlogs, and sends the binlogs downstream
- Drainer supports `relay log`. By the relay log, Drainer ensures that the downstream clusters are in a consistent state.

#### 10.7.1.3 Notes

- In v5.1, the incompatibility between the clustered index feature introduced in v5.0 and TiDB Binlog has been resolved. After you upgrade TiDB Binlog and TiDB Server to v5.1 and enable TiDB Binlog, TiDB will support creating tables with clustered indexes; data insertion, deletion, and update on the created tables with clustered indexes will be replicated to the downstream via TiDB Binlog. When you use TiDB Binlog to replicate the tables with clustered indexes, pay attention to the following:
  - If you have upgraded the cluster to v5.1 from v5.0 by manually controlling the upgrade sequence, make sure that TiDB binlog is upgraded to v5.1 before upgrading the TiDB server to v5.1.
  - It is recommended to configure the system variable `tidb_enable_clustered_index` ↗ to a same value to ensure that the structure of TiDB clustered index tables between the upstream and downstream is consistent.

- TiDB Binlog is incompatible with the following features introduced in TiDB v5.0 and they cannot be used together.
  - **TiDB Clustered Index:** After TiDB Binlog is enabled, TiDB does not allow creating clustered indexes with non-single integer columns as primary keys; data insertion, deletion, and update of the created clustered index tables will not be replicated downstream via TiDB Binlog. If you need to replicate tables with clustered indexes, upgrade your cluster to v5.1 or use [TiCDC](#) instead.
  - TiDB system variable `tidb_enable_async_commit`: After TiDB Binlog is enabled, performance cannot be improved by enabling this option. It is recommended to use [TiCDC](#) instead of TiDB Binlog.
  - TiDB system variable `tidb_enable_1pc`: After TiDB Binlog is enabled, performance cannot be improved by enabling this option. It is recommended to use [TiCDC](#) instead of TiDB Binlog.
- TiDB Binlog is incompatible with the following feature introduced in TiDB v4.0.7 and they cannot be used together:
  - TiDB system variable `tidb_enable_amend_pessimistic_txn`: The two features have compatibility issues. Using them together might cause the issue that TiDB Binlog replicates data inconsistently.
- Drainer supports replicating binlogs to MySQL, TiDB, Kafka or local files. If you need to replicate binlogs to other Drainer unsupported destinations, you can set Drainer to replicate the binlog to Kafka and read the data in Kafka for customized processing according to binlog consumer protocol. See [Binlog Consumer Client User Guide](#).
- To use TiDB Binlog for recovering incremental data, set the config `db-type` to `file` ↞ (local files in the proto buffer format). Drainer converts the binlog to data in the specified [proto buffer format](#) and writes the data to local files. In this way, you can use [Reparo](#) to recover data incrementally.

Pay attention to the value of `db-type`:

- If your TiDB version is earlier than 2.1.9, set `db-type="pb"`.
- If your TiDB version is 2.1.9 or later, set `db-type="file"` or `db-type="pb"`.
- If the downstream is MySQL, MariaDB, or another TiDB cluster, you can use [sync-diff-inspector](#) to verify the data after data replication.

### 10.7.2 TiDB Binlog Tutorial

This tutorial starts with a simple TiDB Binlog deployment with a single node of each component (Placement Driver, TiKV Server, TiDB Server, Pump, and Drainer), set up to push data into a MariaDB Server instance.

This tutorial is targeted toward users who have some familiarity with the [TiDB Architecture](#), who may have already set up a TiDB cluster (not mandatory), and who wants to gain hands-on experience with TiDB Binlog. This tutorial is a good way to “kick the tires” of TiDB Binlog and to familiarize yourself with the concepts of its architecture.

#### Warning:

The instructions to deploy TiDB in this tutorial should **not** be used to deploy TiDB in a production or development setting.

This tutorial assumes you’re using a modern Linux distribution on x86-64. A minimal CentOS 7 installation running in VMware is used in this tutorial for the examples. It’s recommended that you start from a clean install, so that you aren’t impacted by quirks of your existing environment. If you don’t want to use local virtualization, you can easily start a CentOS 7 VM using your cloud service.

#### 10.7.2.1 TiDB Binlog Overview

TiDB Binlog is a solution to collect binary log data from TiDB and provide real-time data backup and replication. It pushes incremental data updates from a TiDB Server cluster into downstream platforms.

You can use TiDB Binlog for incremental backups, to replicate data from one TiDB cluster to another, or to send TiDB updates through Kafka to a downstream platform of your choice.

TiDB Binlog is particularly useful when you migrate data from MySQL or MariaDB to TiDB, in which case you may use the TiDB DM (Data Migration) platform to get data from a MySQL/MariaDB cluster into TiDB, and then use TiDB Binlog to keep a separate, downstream MySQL/MariaDB instance/cluster in sync with your TiDB cluster. TiDB Binlog enables application traffic to TiDB to be pushed to a downstream MySQL or MariaDB instance/cluster, which reduces the risk of a migration to TiDB because you can easily revert the application to MySQL or MariaDB without downtime or data loss.

See [TiDB Binlog Cluster User Guide](#) for more information.

#### 10.7.2.2 Architecture

TiDB Binlog comprises two components: the **Pump** and the **Drainer**. Several Pump nodes make up a pump cluster. Each Pump node connects to TiDB Server instances and receives updates made to each of the TiDB Server instances in a cluster. A Drainer connects to the Pump cluster and transforms the received updates into the correct format for a particular downstream destination, for example, Kafka, another TiDB Cluster or a MySQL/MariaDB server.

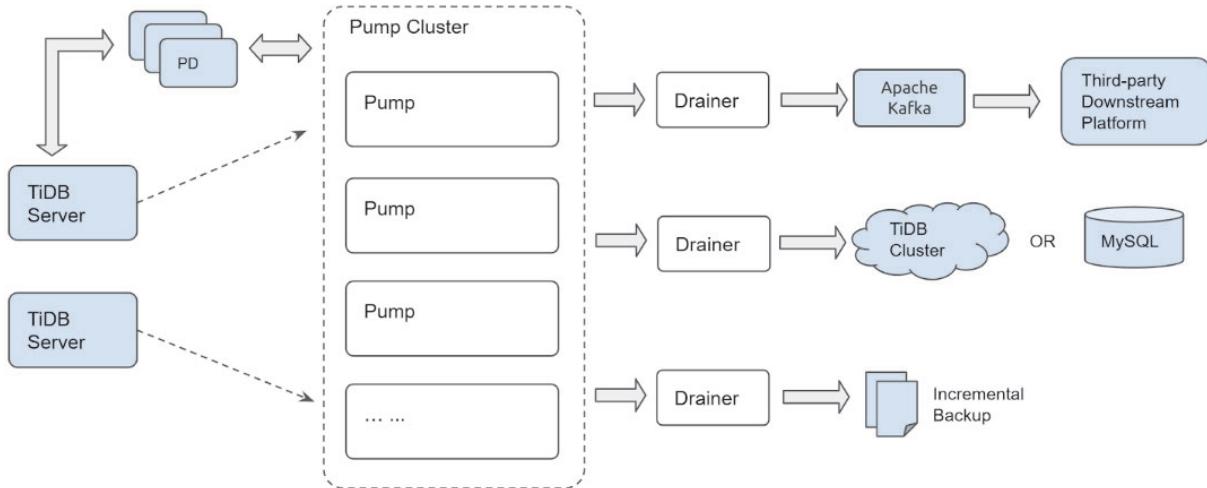


Figure 143: TiDB-Binlog architecture

The clustered architecture of Pump ensures that updates won't be lost as new TiDB Server instances join or leave the TiDB Cluster or Pump nodes join or leave the Pump cluster.

#### 10.7.2.3 Installation

We're using MariaDB Server in this case instead of MySQL Server because RHEL/CentOS 7 includes MariaDB Server in their default package repositories. We'll need the client as well as the server for later use. Let's install them now:

```
sudo yum install -y mariadb-server
```

```
curl -L https://download.pingcap.org/tidb-latest-linux-amd64.tar.gz | tar
  ↪ xzf -
cd tidb-latest-linux-amd64
```

Expected output:

```
[kolbe@localhost ~]$ curl -LO https://download.pingcap.org/tidb-latest-linux-
  ↪ -amd64.tar.gz | tar xzf -
% Total    % Received % Xferd Average Speed Time  Time     Current
               Dload Upload Total Spent   Left Speed
100 368M 100 368M    0     0 8394k      0 0:00:44 0:00:44 --:--:-- 11.1M
[kolbe@localhost ~]$ cd tidb-latest-linux-amd64
[kolbe@localhost tidb-latest-linux-amd64]$
```

#### 10.7.2.4 Configuration

Now we'll start a simple TiDB cluster, with a single instance for each of `pd-server`, `tikv-server`, and `tidb-server`.

Populate the config files using:

```
printf > pd.toml %s\n 'log-file="pd.log"' 'data-dir="pd.data"'
printf > tikv.toml %s\n 'log-file="tikv.log"' '[storage]' 'data-dir="tikv.
    ↪ data"' '[pd]' 'endpoints=[\"127.0.0.1:2379\"]' '[rocksdb]' max-open-
    ↪ files=1024 '[raftdb]' max-open-files=1024
printf > pump.toml %s\n 'log-file="pump.log"' 'data-dir="pump.data"' 'addr
    ↪ ="127.0.0.1:8250"' 'advertise-addr="127.0.0.1:8250"' 'pd-urls="http
    ↪ ://127.0.0.1:2379"'
printf > tidb.toml %s\n 'store="tikv"' 'path="127.0.0.1:2379"' '[log.file
    ↪ ]' 'filename="tidb.log"' '[binlog]' 'enable=true'
printf > drainer.toml %s\n 'log-file="drainer.log"' '[syncer]' 'db-type="
    ↪ mysql"' '[syncer.to]' 'host="127.0.0.1"' 'user="root"' 'password=""'
    ↪ 'port=3306'
```

Use the following commands to see the configuration details:

```
for f in *.toml; do echo "$f:"; cat "$f"; echo; done
```

Expected output:

```
drainer.toml:
log-file="drainer.log"
[syncer]
db-type="mysql"
[syncer.to]
host="127.0.0.1"
user="root"
password=""
port=3306

pd.toml:
log-file="pd.log"
data-dir="pd.data"

pump.toml:
log-file="pump.log"
data-dir="pump.data"
addr="127.0.0.1:8250"
advertise-addr="127.0.0.1:8250"
pd-urls="http://127.0.0.1:2379"

tidb.toml:
```

```

store="tikv"
path="127.0.0.1:2379"
[log.file]
filename="tidb.log"
[binlog]
enable=true

tikv.toml:
log-file="tikv.log"
[storage]
data-dir="tikv.data"
[pd]
endpoints=["127.0.0.1:2379"]
[rocksdb]
max-open-files=1024
[raftdb]
max-open-files=1024

```

#### 10.7.2.5 Bootstrapping

Now we can start each component. This is best done in a specific order - firstly the Placement Driver (PD), then TiKV Server, then Pump (because TiDB must connect to the Pump service to send the binary log), and finally the TiDB Server.

Start all the services using:

```

./bin/pd-server --config=pd.toml &>pd.out &
./bin/tikv-server --config=tikv.toml &>tikv.out &
./bin/pump --config=pump.toml &>pump.out &
sleep 3
./bin/tidb-server --config=tidb.toml &>tidb.out &

```

Expected output:

```

[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/pd-server --config=pd.toml
↪ &>pd.out &
[1] 20935
[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/tikv-server --config=tikv.
↪ toml &>tikv.out &
[2] 20944
[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/pump --config=pump.toml &>
↪ pump.out &
[3] 21050
[kolbe@localhost tidb-latest-linux-amd64]$ sleep 3
[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/tidb-server --config=tidb.
↪ toml &>tidb.out &

```

[4] 21058

If you execute `jobs`, you should see a list of running daemons:

```
[kolbe@localhost tidb-latest-linux-amd64]$ jobs
[1]  Running                  ./bin/pd-server --config=pd.toml &>pd.out &
[2]  Running                  ./bin/tikv-server --config=tikv.toml &>tikv.out &
[3]- Running                  ./bin/pump --config=pump.toml &>pump.out &
[4]+ Running                  ./bin/tidb-server --config=tidb.toml &>tidb.out &
```

If one of the services has failed to start (if you see “Exit 1” instead of “Running”, for example), try to restart that individual service.

#### 10.7.2.6 Connecting

You should have all 4 components of our TiDB Cluster running now, and you can now connect to the TiDB Server on port 4000 using the MariaDB/MySQL command-line client:

```
mysql -h 127.0.0.1 -P 4000 -u root -e 'select tidb_version()\G'
```

Expected output:

```
[kolbe@localhost tidb-latest-linux-amd64]$ mysql -h 127.0.0.1 -P 4000 -u
→ root -e 'select tidb_version()\G'
***** 1. row *****
tidb_version(): Release Version: v3.0.0-beta.1-154-gd5afff70c
Git Commit Hash: d5afff70cdd825d5fab125c8e52e686cc5fb9a6e
Git Branch: master
UTC Build Time: 2019-04-24 03:10:00
GoVersion: go version go1.12 linux/amd64
Race Enabled: false
TIKV Min Version: 2.1.0-alpha.1-ff3dd160846b7d1aed9079c389fc188f7f5ea13e
Check Table Before Drop: false
```

At this point we have a TiDB Cluster running, and we have pump reading binary logs from the cluster and storing them as relay logs in its data directory. The next step is to start a MariaDB server that drainer can write to.

Start drainer using:

```
sudo systemctl start mariadb
./bin/drainer --config=drainer.toml &>drainer.out &
```

If you are using an operating system that makes it easier to install MySQL server, that’s also OK. Just make sure it’s listening on port 3306 and that you can either connect to it as user “root” with an empty password, or adjust drainer.toml as necessary.

```
mysql -h 127.0.0.1 -P 3306 -u root
```

```
show databases;
```

Expected output:

```
[kolbe@localhost ~]$ mysql -h 127.0.0.1 -P 3306 -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 20
Server version: 5.5.60-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
→ statement.

MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| test           |
| tidb_binlog    |
+-----+
5 rows in set (0.01 sec)
```

Here we can already see the `tidb_binlog` database, which contains the `checkpoint` table used by `drainer` to record up to what point binary logs from the TiDB cluster have been applied.

```
MariaDB [tidb_binlog]> use tidb_binlog;
Database changed
MariaDB [tidb_binlog]> select * from checkpoint;
+-----+
| clusterID      | checkPoint          |
+-----+
| 6678715361817107733 | {"commitTS":407637466476445697, "ts-map":{}} |
+-----+
1 row in set (0.00 sec)
```

Now, let's open another client connection to the TiDB server, so that we can create a table and insert some rows into it. (It's recommended that you do this under a GNU screen so you can keep multiple clients open at the same time.)

```
mysql -h 127.0.0.1 -P 4000 --prompt='TiDB [\d]> ' -u root
```

```
create database tidbtest;
use tidbtest;
create table t1 (id int unsigned not null AUTO_INCREMENT primary key);
insert into t1 () values (),(),(),(),();
select * from t1;
```

Expected output:

```
TiDB [(none)]> create database tidbtest;
Query OK, 0 rows affected (0.12 sec)

TiDB [(none)]> use tidbtest;
Database changed
TiDB [tidbtest]> create table t1 (id int unsigned not null AUTO_INCREMENT
    ↪ primary key);
Query OK, 0 rows affected (0.11 sec)

TiDB [tidbtest]> insert into t1 () values (),(),(),(),();
Query OK, 5 rows affected (0.01 sec)
Records: 5 Duplicates: 0 Warnings: 0

TiDB [tidbtest]> select * from t1;
+---+
| id |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+---+
5 rows in set (0.00 sec)
```

Switching back to the MariaDB client, we should find the new database, new table, and the newly inserted rows:

```
use tidbtest;
show tables;
select * from t1;
```

Expected output:

```
MariaDB [(none)]> use tidbtest;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```

Database changed
MariaDB [tidbtest]> show tables;
+-----+
| Tables_in_tidbtest |
+-----+
| t1           |
+-----+
1 row in set (0.00 sec)

MariaDB [tidbtest]> select * from t1;
+---+
| id |
+---+
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
+---+
5 rows in set (0.00 sec)

```

You should see the same rows that you inserted into TiDB when querying the MariaDB server. Congratulations! You've just set up TiDB Binlog!

#### 10.7.2.7 binlogctl

Information about Pumps and Drainers that have joined the cluster is stored in PD. You can use the binlogctl tool query and manipulate information about their states. See [binlogctl guide](#) for more information.

Use binlogctl to get a view of the current status of Pumps and Drainers in the cluster:

```

./bin/binlogctl -cmd drainers
./bin/binlogctl -cmd pumps

```

Expected output:

```

[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/binlogctl -cmd drainers
[2019/04/11 17:44:10.861 -04:00] [INFO] [nodes.go:47] ["query node"] [type=
    ↵ drainer] [node="{NodeID: localhost.localdomain:8249, Addr:
    ↵ 192.168.236.128:8249, State: online, MaxCommitTS: 407638907719778305,
    ↵ UpdateTime: 2019-04-11 17:44:10 -0400 EDT}"]

[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/binlogctl -cmd pumps
[2019/04/11 17:44:13.904 -04:00] [INFO] [nodes.go:47] ["query node"] [type=
    ↵ pump] [node="{NodeID: localhost.localdomain:8250, Addr:

```

```
↪ 192.168.236.128:8250, State: online, MaxCommitTS: 407638914024079361,
↪ UpdateTime: 2019-04-11 17:44:13 -0400 EDT}"]
```

If you kill a Drainer, the cluster puts it in the “paused” state, which means that the cluster expects it to rejoin:

```
pkill drainer
./bin/binlogctl -cmd drainers
```

Expected output:

```
[kolbe@localhost tidb-latest-linux-amd64]$ pkill drainer
[kolbe@localhost tidb-latest-linux-amd64]$ ./bin/binlogctl -cmd drainers
[2019/04/11 17:44:22.640 -04:00] [INFO] [nodes.go:47] ["query node"] [type=
↪ drainer] [node="{NodeID: localhost.localdomain:8249, Addr:
↪ 192.168.236.128:8249, State: paused, MaxCommitTS: 407638915597467649,
↪ UpdateTime: 2019-04-11 17:44:18 -0400 EDT}"]
```

You can use “NodeIDs” with `binlogctl` to control individual nodes. In this case, the NodeID of the drainer is “localhost.localdomain:8249” and the NodeID of the Pump is “localhost.localdomain:8250”.

The main use of `binlogctl` in this tutorial is likely to be in the event of a cluster restart. If you end all processes in the TiDB cluster and try to restart them (not including the downstream MySQL/MariaDB server or Drainer), Pump will refuse to start because it cannot contact Drainer and believe that Drainer is still “online”.

There are 3 solutions to this issue:

- Stop Drainer using `binlogctl` instead of killing the process:

```
./bin/binlogctl --pd-urls=http://127.0.0.1:2379 --cmd=drainers
./bin/binlogctl --pd-urls=http://127.0.0.1:2379 --cmd=offline-drainer
↪ --node-id=localhost.localdomain:8249
```

- Start Drainer *before* starting Pump.
- Use `binlogctl` after starting PD (but before starting Drainer and Pump) to update the state of the paused Drainer:

```
./bin/binlogctl --pd-urls=http://127.0.0.1:2379 --cmd=update-drainer --
↪ node-id=localhost.localdomain:8249 --state=offline
```

### 10.7.2.8 Cleanup

To stop the TiDB cluster and TiDB Binlog processes, you can execute `pkill -P $$` in the shell where you started all the processes that form the cluster (pd-server, tikv-server, pump, tidb-server, drainer). To give each component enough time to shut down cleanly, it’s helpful to stop them in a particular order:

```
for p in tidb-server drainer pump tikv-server pd-server; do pkill "$p";
↪ sleep 1; done
```

Expected output:

```
kolbe@localhost tidb-latest-linux-amd64]$ for p in tidb-server drainer pump
↪ tikv-server pd-server; do pkill "$p"; sleep 1; done
[4]- Done                  ./bin/tidb-server --config=tidb.toml &>tidb.out
[5]+ Done                  ./bin/drainer --config=drainer.toml &>drainer.out
[3]+ Done                  ./bin/pump --config=pump.toml &>pump.out
[2]+ Done                  ./bin/tikv-server --config=tikv.toml &>tikv.out
[1]+ Done                  ./bin/pd-server --config=pd.toml &>pd.out
```

If you wish to restart the cluster after all services exit, use the same commands you ran originally to start the services. As discussed in the [binlogctl](#) section above, you'll need to start drainer before pump, and pump before tidb-server.

```
./bin/pd-server --config=pd.toml &>pd.out &
./bin/tikv-server --config=tikv.toml &>tikv.out &
./bin/drainer --config=drainer.toml &>drainer.out &
sleep 3
./bin/pump --config=pump.toml &>pump.out &
sleep 3
./bin/tidb-server --config=tidb.toml &>tidb.out &
```

If any of the components fail to start, try to restart the failed individual component(s).

#### 10.7.2.9 Conclusion

In this tutorial, we've set up TiDB Binlog to replicate from a TiDB cluster to a downstream MariaDB server, using a cluster with a single Pump and a single Drainer. As we've seen, TiDB Binlog is a comprehensive platform for capturing and processing changes to a TiDB cluster.

In a more robust development, testing, or production deployment, you'd have multiple TiDB servers for high availability and scaling purposes, and you'd use multiple Pump instances to ensure that application traffic to TiDB server instances is unaffected by problems in the Pump cluster. You may also use additional Drainer instances to push updates to different downstream platforms or to implement incremental backups.

#### 10.7.3 TiDB Binlog Cluster Deployment

This document describes how to [deploy TiDB Binlog using a Binary package](#).

### 10.7.3.1 Hardware requirements

Pump and Drainer are deployed and operate on 64-bit universal hardware server platforms with Intel x86-64 architecture.

In environments of development, testing and production, the requirements on server hardware are as follows:

Service	The Number of Servers	CPU	Disk	Memory
Pump	3	8 core+	SSD, 200 GB+	16G
Drainer	1	8 core+	SAS, 100 GB+ (If binlogs are output as local files, the disk size depends on how long these files are retained.)	16G

### 10.7.3.2 Deploy TiDB Binlog using TiUP

It is recommended to deploy TiDB Binlog using TiUP. To do that, when deploying TiDB using TiUP, you need to add the node information of `drainer` and `pump` of TiDB Binlog in [TiDB Binlog Deployment Topology](#). For detailed deployment information, refer to [Deploy a TiDB Cluster Using TiUP](#).

### 10.7.3.3 Deploy TiDB Binlog using a Binary package

#### 10.7.3.3.1 Download the official Binary package

Run the following commands to download the packages:

```
version="latest" for nightly builds &&
wget https://download.pingcap.org/tidb-latest-linux-amd64.{tar.gz,sha256}
```

Check the file integrity. If the result is OK, the file is correct.

```
sha256sum -c tidb-latest-linux-amd64.sha256
```

For TiDB v2.1.0 GA or later versions, Pump and Drainer are already included in the TiDB download package. For other TiDB versions, you need to download Pump and Drainer separately using the following command:

```
wget https://download.pingcap.org/tidb-binlog-$version-linux-amd64.{tar.gz,
↪ sha256}
```

Check the file integrity. If the result is OK, the file is correct.

```
sha256sum -c tidb-binlog-$version-linux-amd64.sha256
```

#### 10.7.3.3.2 The usage example

Assuming that you have three PD nodes, one TiDB node, two Pump nodes, and one Drainer node, the information of each node is as follows:

Node	IP
TiDB	192.168.0.10
PD1	192.168.0.16
PD2	192.168.0.15
PD3	192.168.0.14
Pump	192.168.0.11
Pump	192.168.0.12
Drainer	192.168.0.13

The following part shows how to use Pump and Drainer based on the nodes above.

1. Deploy Pump using the binary.

- To view the command line parameters of Pump, execute `./bin/pump -help`:

```
Usage of Pump:
-L string
    the output information level of logs: debug, info, warn, error,
    ↪ fatal ("info" by default)
-V
    the print version information
-addr string
    the RPC address through which Pump provides the service (-addr=
    ↪ "192.168.0.11:8250")
-advertise-addr string
    the RPC address through which Pump provides the external
    ↪ service (-advertise-addr="192.168.0.11:8250")
-config string
    the path of the configuration file. If you specify the
    ↪ configuration file, Pump reads the configuration in the
    ↪ configuration file first. If the corresponding
    ↪ configuration also exists in the command line parameters,
    ↪ Pump uses the configuration of the command line
    ↪ parameters to cover that of the configuration file.
-data-dir string
```

```

    the path where the Pump data is stored
-gc int
    the number of days to retain the data in Pump ("7" by default)
-heartbeat-interval int
    the interval of the heartbeats Pump sends to PD (in seconds)
-log-file string
    the file path of logs
-log-rotate string
    the switch frequency of logs (hour/day)
-metrics-addr string
    the Prometheus Pushgateway address. If not set, it is forbidden
        ↪ to report the monitoring metrics.
-metrics-interval int
    the report frequency of the monitoring metrics ("15" by default
        ↪ , in seconds)
-node-id string
    the unique ID of a Pump node. If you do not specify this ID,
        ↪ the system automatically generates an ID based on the
        ↪ host name and listening port.
-pd-urls string
    the address of the PD cluster nodes (-pd-urls="http
        ↪ ://192.168.0.16:2379,http://192.168.0.15:2379,http
        ↪ ://192.168.0.14:2379")
-fake-binlog-interval int
    the frequency at which a Pump node generates fake binlog ("3"
        ↪ by default, in seconds)

```

- Taking deploying Pump on “192.168.0.11” as an example, the Pump configuration file is as follows:

```

# Pump Configuration

# the bound address of Pump
addr = "192.168.0.11:8250"

# the address through which Pump provides the service
advertise-addr = "192.168.0.11:8250"

# the number of days to retain the data in Pump ("7" by default)
gc = 7

# the directory where the Pump data is stored
data-dir = "data.pump"

# the interval of the heartbeats Pump sends to PD (in seconds)

```

```

heartbeat-interval = 2

# the address of the PD cluster nodes (each separated by a comma
#   ↪ with no whitespace)
pd-urls = "http://192.168.0.16:2379,http://192.168.0.15:2379,http
#   ↪ ://192.168.0.14:2379"

# [security]
# This section is generally commented out if no special security
#   ↪ settings are required.
# The file path containing a list of trusted SSL CAs connected to
#   ↪ the cluster.
# ssl-ca = "/path/to/ca.pem"
# The path to the X509 certificate in PEM format that is connected
#   ↪ to the cluster.
# ssl-cert = "/path/to/drainer.pem"
# The path to the X509 key in PEM format that is connected to the
#   ↪ cluster.
# ssl-key = "/path/to/drainer-key.pem"

# [storage]
# Set to true (by default) to guarantee reliability by ensuring
#   ↪ binlog data is flushed to the disk
# sync-log = true

# When the available disk capacity is less than the set value, Pump
#   ↪ stops writing data.
# 42 MB -> 42000000, 42 mib -> 44040192
# default: 10 gib
# stop-write-at-available-space = "10 gib"
# The LSM DB settings embedded in Pump. Unless you know this part
#   ↪ well, it is usually commented out.
# [storage.kv]
# block-cache-capacity = 8388608
# block-restart-interval = 16
# block-size = 4096
# compaction-L0-trigger = 8
# compaction-table-size = 67108864
# compaction-total-size = 536870912
# compaction-total-size-multiplier = 8.0
# write-buffer = 67108864
# write-L0-pause-trigger = 24
# write-L0-slowdown-trigger = 17

```

- The example of starting Pump:

```
./bin/pump -config pump.toml
```

If the command line parameters is the same with the configuration file parameters, the values of command line parameters are used.

## 2. Deploy Drainer using binary.

- To view the command line parameters of Drainer, execute `./bin/drainer -help`:

```
Usage of Drainer:
-L string
    the output information level of logs: debug, info, warn, error,
    ↪ fatal ("info" by default)
-V
    the print version information
-addr string
    the address through which Drainer provides the service (-addr="
    ↪ 192.168.0.13:8249")
-c int
    the number of the concurrency of the downstream for replication
    ↪ . The bigger the value, the better throughput performance
    ↪ of the concurrency ("1" by default).
-cache-binlog-count int
    the limit on the number of binlog items in the cache ("8" by
    ↪ default)
    If a large single binlog item in the upstream causes OOM in
    ↪ Drainer, try to lower the value of this parameter to
    ↪ reduce memory usage.
-config string
    the directory of the configuration file. Drainer reads the
    ↪ configuration file first.
    If the corresponding configuration exists in the command line
    ↪ parameters, Drainer uses the configuration of the command
    ↪ line parameters to cover that of the configuration file.
-data-dir string
    the directory where the Drainer data is stored ("data.drainer"
    ↪ by default)
-dest-db-type string
    the downstream service type of Drainer
    The value can be "mysql", "tidb", "kafka", and "file". ("mysql"
    ↪ by default)
-detect-interval int
    the interval of checking the online Pump in PD ("10" by default
    ↪ , in seconds)
-disable-detect
```

```

        whether to disable the conflict monitoring
-disable-dispatch
        whether to disable the SQL feature of splitting a single binlog
        ↵ file. If it is set to "true", each binlog file is
        ↵ restored to a single transaction for replication based on
        ↵ the order of binlogs.
        It is set to "False", when the downstream is MySQL.
-ignore-schemas string
        the db filter list ("INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,
        ↵ mysql,test" by default)
        It does not support the Rename DDL operation on tables of
        ↵ ignore schemas'.
-initial-commit-ts
        If Drainer does not have the related breakpoint information,
        ↵ you can configure the related breakpoint information
        ↵ using this parameter. ("-1" by default)
        If the value of this parameter is '-1', Drainer automatically
        ↵ obtains the latest timestamp from PD.
-log-file string
        the path of the log file
-log-rotate string
        the switch frequency of log files, hour/day
-metrics-addr string
        the Prometheus Pushgateway address
        It it is not set, the monitoring metrics are not reported.
-metrics-interval int
        the report frequency of the monitoring metrics ("15" by default
        ↵ , in seconds)
-node-id string
        the unique ID of a Drainer node. If you do not specify this ID,
        ↵ the system automatically generates an ID based on the
        ↵ host name and listening port.
-pd-urls string
        the address of the PD cluster nodes (-pd-urls="http
        ↵ ://192.168.0.16:2379,http://192.168.0.15:2379,http
        ↵ ://192.168.0.14:2379")
-safe-mode
        Whether to enable safe mode so that data can be written into
        ↵ the downstream MySQL/TiDB repeatedly.
        This mode replaces the `INSERT` statement with the `REPLACE`
        ↵ statement and splits the `UPDATE` statement into `DELETE`
        ↵ plus `REPLACE`.
-txn-batch int
        the number of SQL statements of a transaction which are output
        ↵ to the downstream database ("1" by default)

```

- Taking deploying Drainer on “192.168.0.13” as an example, the Drainer configuration file is as follows:

```
# Drainer Configuration.

# the address through which Drainer provides the service
    ↪ ("192.168.0.13:8249")
addr = "192.168.0.13:8249"

# the address through which Drainer provides the external service
advertise-addr = "192.168.0.13:8249"

# the interval of checking the online Pump in PD ("10" by default,
    ↪ in seconds)
detect-interval = 10

# the directory where the Drainer data is stored "data.drainer" by
    ↪ default)
data-dir = "data.drainer"

# the address of the PD cluster nodes (each separated by a comma
    ↪ with no whitespace)
pd-urls = "http://192.168.0.16:2379,http://192.168.0.15:2379,http
    ↪ ://192.168.0.14:2379"

# the directory of the log file
log-file = "drainer.log"

# Drainer compresses the data when it gets the binlog from Pump.
    ↪ The value can be "gzip". If it is not configured, it will
    ↪ not be compressed
# compressor = "gzip"

# [security]
# This section is generally commented out if no special security
    ↪ settings are required.
# The file path containing a list of trusted SSL CAs connected to
    ↪ the cluster.
# ssl-ca = "/path/to/ca.pem"
# The path to the X509 certificate in PEM format that is connected
    ↪ to the cluster.
# ssl-cert = "/path/to/pump.pem"
# The path to the X509 key in PEM format that is connected to the
    ↪ cluster.
```

```

# ssl-key = "/path/to/pump-key.pem"

# Syncer Configuration
[syncer]
# If the item is set, the sql-mode will be used to parse the DDL
#   ↪ statement.
# If the downstream database is MySQL or TiDB, then the downstream
#   ↪ sql-mode
# is also set to this value.
# sql-mode = "STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION"

# the number of SQL statements of a transaction that are output to
#   ↪ the downstream database ("20" by default)
txn-batch = 20

# the number of the concurrency of the downstream for replication.
#   ↪ The bigger the value,
# the better throughput performance of the concurrency ("16" by
#   ↪ default)
worker-count = 16

# whether to disable the SQL feature of splitting a single binlog
#   ↪ file. If it is set to "true",
# each binlog file is restored to a single transaction for
#   ↪ replication based on the order of binlogs.
# If the downstream service is MySQL, set it to "False".
disable-dispatch = false

# In safe mode, data can be written into the downstream MySQL/TiDB
#   ↪ repeatedly.
# This mode replaces the `INSERT` statement with the `REPLACE`
#   ↪ statement and replaces the `UPDATE` statement with `DELETE`
#   ↪ plus `REPLACE` statements.
safe-mode = false

# the downstream service type of Drainer ("mysql" by default)
# Valid value: "mysql", "tidb", "file", and "kafka".
db-type = "mysql"

# If `commit ts` of the transaction is in the list, the transaction
#   ↪ is filtered and not replicated to the downstream.
ignore-txn-commit-ts = []

# the db filter list ("INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql,
#   ↪ test" by default)

```

```

# Does not support the Rename DDL operation on tables of `ignore
#   schemas`.
ignore-schemas = "INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql"

# `replicate-do-db` has priority over `replicate-do-table`. When
#   they have the same `db` name,
# regular expressions are supported for configuration.
# The regular expression should start with "~".

# replicate-do-db = ["~^b.*","s1"]

# [syncer.relay]
# It saves the directory of the relay log. The relay log is not
#   enabled if the value is empty.
# The configuration only comes to effect if the downstream is TiDB
#   or MySQL.
# log-dir = ""
# the maximum size of each file
# max-file-size = 10485760

# [[syncer.replicate-do-table]]
# db-name ="test"
# tbl-name = "log"

# [[syncer.replicate-do-table]]
# db-name ="test"
# tbl-name = "~^a.*"

# Ignore the replication of some tables
# [[syncer.ignore-table]]
# db-name = "test"
# tbl-name = "log"

# the server parameters of the downstream database when `db-type`
#   is set to "mysql"
[syncer.to]
host = "192.168.0.13"
user = "root"
# If you do not want to set a cleartext `password` in the
#   configuration file, you can create `encrypted_password`
#   using `./binlogctl -cmd encrypt -text string`.
# When you have created an `encrypted_password` that is not empty,
#   the `password` above will be ignored, because `
#   encrypted_password` and `password` cannot take effect at the
#   same time.

```

```

password = ""
encrypted_password = ""
port = 3306

[syncer.to.checkpoint]
# When the checkpoint type is "mysql" or "tidb", this option can be
# enabled to change the database that saves the checkpoint
# schema = "tidb_binlog"
# Currently only the "mysql" and "tidb" checkpoint types are
# → supported
# You can remove the comment tag to control where to save the
# → checkpoint
# The default method of saving the checkpoint for the downstream db
# → -type:
# mysql/tidb -> in the downstream MySQL or TiDB database
# file/kafka -> file in `data-dir`
# type = "mysql"
# host = "127.0.0.1"
# user = "root"
# password = ""
# `encrypted_password` is encrypted using `./binlogctl -cmd encrypt
# → -text string`.
# When `encrypted_password` is not empty, the `password` above will
# → be ignored.
# encrypted_password = ""
# port = 3306

# the directory where the binlog file is stored when `db-type` is
# → set to `file`
# [syncer.to]
# dir = "data.drainer"

# the Kafka configuration when `db-type` is set to "kafka"
# [syncer.to]
# only one of kafka-addrs and zookeeper-addrs is needed. If both
# → are present, the program gives priority
# to the kafka address in zookeeper
# zookeeper-addrs = "127.0.0.1:2181"
# kafka-addrs = "127.0.0.1:9092"
# kafka-version = "0.8.2.0"
# The maximum number of messages (number of binlogs) in a broker
# → request. If it is left blank or a value smaller than 0 is
# → configured, the default value 1024 is used.
# kafka-max-messages = 1024
# The maximum size of a broker request (unit: byte). The default

```

```

    ↪ value is 1 GiB and the maximum value is 2 GiB.
# kafka-max-message-size = 1073741824

# the topic name of the Kafka cluster that saves the binlog data.
    ↪ The default value is <cluster-id>_obinlog.
# To run multiple Drainers to replicate data to the same Kafka
    ↪ cluster, you need to set different `topic-name`'s for each
    ↪ Drainer.
# topic-name = ""

```

- Starting Drainer:

**Note:**

If the downstream is MySQL/TiDB, to guarantee the data integrity, you need to obtain the `initial-commit-ts` value and make a full backup of the data and restore the data before the initial start of Drainer.

When Drainer is started for the first time, use the `initial-commit-ts` parameter.

```
./bin/drainer -config drainer.toml -initial-commit-ts {initial-
    ↪ commit-ts}
```

If the command line parameter and the configuration file parameter are the same, the parameter value in the command line is used.

3. Starting TiDB server:

- After starting Pump and Drainer, start TiDB server with binlog enabled by adding this section to your config file for TiDB server:

```
[binlog]
enable=true
```

- TiDB server will obtain the addresses of registered Pumps from PD and will stream data to all of them. If there are no registered Pump instances, TiDB server will refuse to start or will block starting until a Pump instance comes online.

**Note:**

- When TiDB is running, you need to guarantee that at least one Pump is running normally.

- To enable the TiDB Binlog service in TiDB server, use the `-enable-binlog` startup parameter in TiDB, or add `enable=true` to the `binlog` section of the TiDB server configuration file.
- Make sure that the TiDB Binlog service is enabled in all TiDB instances in a same cluster, otherwise upstream and downstream data inconsistency might occur during data replication. If you want to temporarily run a TiDB instance where the TiDB Binlog service is not enabled, set `run_ddl=false` in the TiDB configuration file.
- Drainer does not support the `rename` DDL operation on the table of `ignore schemas` (the schemas in the filter list).
- If you want to start Drainer in an existing TiDB cluster, generally you need to make a full backup of the cluster data, obtain `snapshot timestamp`, import the data to the target database, and then start Drainer to replicate the incremental data from the corresponding `snapshot timestamp`.
- When the downstream database is TiDB or MySQL, ensure that the `sql_mode` in the upstream and downstream databases are consistent. In other words, the `sql_mode` should be the same when each SQL statement is executed in the upstream and replicated to the downstream. You can execute the `select @@sql_mode;` statement in the upstream and downstream respectively to compare `sql_mode`.
- When a DDL statement is supported in the upstream but incompatible with the downstream, Drainer fails to replicate data. An example is to replicate the `CREATE TABLE t1(a INT)ROW_FORMAT=FIXED;` statement when the downstream database MySQL uses the InnoDB engine. In this case, you can configure `skipping transactions` in Drainer, and manually execute compatible statements in the downstream database.

#### 10.7.4 TiDB Binlog Cluster Operations

This document introduces the following TiDB Binlog cluster operations:

- The state of a Pump and Drainer nodes
- Starting or exiting a Pump or Drainer process
- Managing the TiDB Binlog cluster by using the `binlogctl` tool or by directly performing SQL operations in TiDB

##### 10.7.4.1 Pump or Drainer state

Pump or Drainer state description:

- `online`: running normally

- **pausing**: in the pausing process
- **paused**: has been stopped
- **closing**: in the offline process
- **offline**: has been offline

**Note:**

The state information of a Pump or Drainer node is maintained by the service itself and is regularly updated to the Placement Driver (PD).

#### 10.7.4.2 Starting and exiting a Pump or Drainer process

##### 10.7.4.2.1 Pump

- Starting: When started, the Pump node notifies all Drainer nodes in the **online** state. If the notification is successful, the Pump node sets its state to **online**. Otherwise, the Pump node reports an error, sets its state to **paused** and exits the process.
- Exiting: The Pump node enters the **paused** or **offline** state before the process is exited normally; if the process is exited abnormally (caused by the `kill -9` command, process panic, crash), the node is still in the **online** state.
  - Pause: You can pause a Pump process by using the `kill` command (not `kill -9 ↵`), pressing Ctrl+C or using the `pause-pump` command in the binlogctl tool. After receiving the pause instruction, the Pump node sets its state to **pausing**, stops receiving binlog write requests and stops providing binlog data to Drainer nodes. After all threads are safely exited, the Pump node updates its state to **paused** and exits the process.
  - Offline: You can close a Pump process only by using the `offline-pump` command in the binlogctl tool. After receiving the offline instruction, the Pump node sets its state to **closing** and stops receiving the binlog write requests. The Pump node continues providing binlog to Drainer nodes until all binlog data is consumed by Drainer nodes. Then, the Pump node sets its state to **offline** and exits the process.

##### 10.7.4.2.2 Drainer

- Starting: When started, the Drainer node sets its state to **online** and tries to pull binlogs from all Pump nodes which are not in the **offline** state. If it fails to get the binlogs, it keeps trying.

- Exiting: The Drainer node enters the `paused` or `offline` state before the process is exited normally; if the process is exited abnormally (caused by `kill -9`, process panic, crash), the Drainer node is still in the `online` state.
  - Pause: You can pause a Drainer process by using the `kill` command (not `kill -9`), pressing Ctrl+C or using the `pause-drainer` command in the binlogctl tool. After receiving the pause instruction, the Drainer node sets its state to `pausing` and stops pulling binlogs from Pump nodes. After all threads are safely exited, the Drainer node sets its state to `paused` and exits the process.
  - Offline: You can close a Drainer process only by using the `offline-drainer` command in the binlogctl tool. After receiving the offline instruction, the Drainer node sets its state to `closing` and stops pulling binlogs from Pump nodes. After all threads are safely exited, the Drainer node updates its state to `offline` and exits the process.

For how to pause, close, check, and modify the state of Drainer, see the [binlogctl guide](#).

#### 10.7.4.3 Use binlogctl to manage Pump/Drainer

`binlogctl` is an operations tool for TiDB Binlog with the following features:

- Checking the state of Pump or Drainer
- Pausing or closing Pump or Drainer
- Handling the abnormal state of Pump or Drainer

For detailed usage of `binlogctl`, refer to [binlogctl overview](#).

#### 10.7.4.4 Use SQL statements to manage Pump or Drainer

To view or modify binlog related states, execute corresponding SQL statements in TiDB.

- Check whether binlog is enabled:

```
show variables like "log_bin";
```

Variable_name	Value
log_bin	0

When the Value is 0, binlog is enabled. When the Value is 1, binlog is disabled.

- Check the status of all the Pump or Drainer nodes:

```
show pump status;
```

NodeID	Address	State	Max_Commit_Ts	Update_Time
pump1	127.0.0.1:8250	Online	408553768673342237	2019-05-01 → 00:00:01
pump2	127.0.0.2:8250	Online	408553768673342335	2019-05-01 → 00:00:02

```
show drainer status;
```

NodeID	Address	State	Max_Commit_Ts	Update_Time
drainer1	127.0.0.3:8249	Online	408553768673342532	2019-05-01 → 00:00:03
drainer2	127.0.0.4:8249	Online	408553768673345531	2019-05-01 → 00:00:04

- Modify the state of a Pump or Drainer node in abnormal situations

```
change pump to node_state ='paused' for node_id 'pump1';
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
change drainer to node_state ='paused' for node_id 'drainer1';
```

```
Query OK, 0 rows affected (0.01 sec)
```

Executing the above SQL statements works the same as the `update-pump` or `update -drainer` commands in binlogctl. Use the above SQL statements **only** when the Pump or Drainer node is in abnormal situations.

**Note:**

- Checking whether binlog is enabled and the running status of Pump or Drainer is supported in TiDB v2.1.7 and later versions.
- Modifying the status of Pump or Drainer is supported in TiDB v3.0.0-rc.1 and later versions. This feature only supports modifying the status of Pump or Drainer nodes stored in PD. To pause or close the node, use the `binlogctl` tool.

## 10.7.5 TiDB Binlog Configuration File

This document introduces the configuration items of TiDB Binlog.

### 10.7.5.1 Pump

This section introduces the configuration items of Pump. For the example of a complete Pump configuration file, see [Pump Configuration](#).

#### 10.7.5.1.1 addr

- Specifies the listening address of HTTP API in the format of `host:port`.
- Default value: `127.0.0.1:8250`

#### 10.7.5.1.2 advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of `host:port`.
- Default value: `127.0.0.1:8250`

#### 10.7.5.1.3 socket

- The Unix socket address that HTTP API listens to.
- Default value: “”

#### 10.7.5.1.4 pd-urls

- Specifies the comma-separated list of PD URLs. If multiple addresses are specified, when the PD client fails to connect to one address, it automatically tries to connect to another address.
- Default value: `http://127.0.0.1:2379`

#### 10.7.5.1.5 `data-dir`

- Specifies the directory where binlogs and their indexes are stored locally.
- Default value: `data.pump`

#### 10.7.5.1.6 `heartbeat-interval`

- Specifies the heartbeat interval (in seconds) at which the latest status is reported to PD.
- Default value: 2

#### 10.7.5.1.7 `gen-binlog-interval`

- Specifies the interval (in seconds) at which data is written into fake binlog.
- Default value: 3

#### 10.7.5.1.8 `gc`

- Specifies the number of days (integer) that binlogs can be stored locally. Binlogs stored longer than the specified number of days are automatically deleted.
- Default value: 7

#### 10.7.5.1.9 `log-file`

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: “”

#### 10.7.5.1.10 `log-level`

- Specifies the log level.
- Default value: `info`

#### 10.7.5.1.11 `node-id`

- Specifies the Pump node ID. With this ID, this Pump process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8250`.

### 10.7.5.1.12 security

This section introduces configuration items related to security.

ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, `/path/to/ca.pem`.
- Default value: `""`

ssl-cert

- Specifies the path of the X509 certificate file encoded in the Privacy Enhanced Mail (PEM) format. For example, `/path/to/pump.pem`.
- Default value: `""`

ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, `/path/to/pump-key.pem`.
- Default value: `""`

### 10.7.5.1.13 storage

This section introduces configuration items related to storage.

sync-log

- Specifies whether to use `fsync` after each **batch** write to binlog to ensure data safety.
- Default value: `true`

kv\_chan\_cap

- Specifies the number of write requests that the buffer can store before Pump receives these requests.
- Default value: 1048576 (that is, 2 to the power of 20)

slow\_write\_threshold

- The threshold (in seconds). If it takes longer to write a single binlog file than this specified threshold, the write is considered slow write and "take a long time to  $\hookrightarrow$  write binlog" is output in the log.
- Default value: 1

## stop-write-at-available-space

- Binlog write requests is no longer accepted when the available storage space is below this specified value. You can use the format such as 900 MB, 5 GB, and 12 GiB to specify the storage space. If there is more than one Pump node in the cluster, when a Pump node refuses a write request because of the insufficient space, TiDB will automatically write binlogs to other Pump nodes.
- Default value: 10 GiB

## kv

Currently the storage of Pump is implemented based on [GoLevelDB](#). Under `storage` there is also a `kv` subgroup that is used to adjust the GoLevel configuration. The supported configuration items are shown as below:

- `block-cache-capacity`
- `block-restart-interval`
- `block-size`
- `compaction-L0-trigger`
- `compaction-table-size`
- `compaction-total-size`
- `compaction-total-size-multiplier`
- `write-buffer`
- `write-L0-pause-trigger`
- `write-L0-slowdown-trigger`

For the detailed description of the above items, see [GoLevelDB Document](#).

### 10.7.5.2 Drainer

This section introduces the configuration items of Drainer. For the example of a complete Drainer configuration file, see [Drainer Configuration](#)

#### 10.7.5.2.1 addr

- Specifies the listening address of HTTP API in the format of `host:port`.
- Default value: `127.0.0.1:8249`

#### 10.7.5.2.2 advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of `host:port`.
- Default value: `127.0.0.1:8249`

#### 10.7.5.2.3 log-file

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: “”

#### 10.7.5.2.4 log-level

- Specifies the log level.
- Default value: `info`

#### 10.7.5.2.5 node-id

- Specifies the Drainer node ID. With this ID, this Drainer process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8249`.

#### 10.7.5.2.6 data-dir

- Specifies the directory used to store files that need to be saved during Drainer operation.
- Default value: `data.drainer`

#### 10.7.5.2.7 detect-interval

- Specifies the interval (in seconds) at which PD updates the Pump information.
- Default value: 5

#### 10.7.5.2.8 pd-urls

- The comma-separated list of PD URLs. If multiple addresses are specified, the PD client will automatically attempt to connect to another address if an error occurs when connecting to one address.
- Default value: `http://127.0.0.1:2379`

#### 10.7.5.2.9 initial-commit-ts

- Specifies from which commit timestamp of the transaction the replication process starts. This configuration is applicable only to the Drainer node that is in the replication process for the first time. If a checkpoint already exists in the downstream, the replication will be performed according to the time recorded in the checkpoint.

- commit\_ts (commit timestamp) is a specific point in time for **transaction** commits in TiDB. It is a globally unique and increasing timestamp from PD as the unique ID of the current transaction. You can get the **initial-commit-ts** configuration in the following typical ways:
  - If BR is used, you can get **initial-commit-ts** from the backup TS recorded in the metadata backed up by BR (backupmeta).
  - If Dumpling is used, you can get **initial-commit-ts** from the Pos recorded in the metadata backed up by Dumpling (metadata),
  - If PD Control is used, **initial-commit-ts** is in the output of the **tso** command.
- Default value: -1. Drainer will get a new timestamp from PD as the starting time, which means that the replication process starts from the current time.

#### 10.7.5.2.10 synced-check-time

- You can access the **/status** path through the HTTP API to query the status of Drainer replication. **synced-check-time** specifies how many minutes from the last successful replication is considered as **synced**, that is, the replication is complete.
- Default value: 5

#### 10.7.5.2.11 compressor

- Specifies the compression algorithm used for data transfer between Pump and Drainer. Currently only the **gzip** algorithm is supported.
- Default value: "", which means no compression.

#### 10.7.5.2.12 security

This section introduces configuration items related to security.

##### ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, **/path/to/ca.pem**.
- Default value: ""

##### ssl-cert

- Specifies the path of the X509 certificate file encoded in the PEM format. For example, **/path/to/drainer.pem**.
- Default value: ""

##### ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, **/path/to/pump-key.pem**.
- Default value: ""

#### 10.7.5.2.13 syncer

The `syncer` section includes configuration items related to the downstream.

db-type

Currently, the following downstream types are supported:

- `mysql`
- `tidb`
- `kafka`
- `file`

Default value: `mysql`

sql-mode

- Specifies the SQL mode when the downstream is the `mysql` or `tidb` type. If there is more than one mode, use commas to separate them.
- Default value: “”

ignore-txn-commit-ts

- Specifies the commit timestamp at which the binlog is ignored, such as `[416815754209656834, ↪ 421349811963822081]`.
- Default value: `[]`

ignore-schemas

- Specifies the database to be ignored during replication. If there is more than one database to be ignored, use commas to separate them. If all changes in a binlog file are filtered, the whole binlog file is ignored.
- Default value: `INFORMATION_SCHEMA, PERFORMANCE_SCHEMA, mysql`

ignore-table

Ignores the specified table changes during replication. You can specify multiple tables to be ignored in the `toml` file. For example:

```
[[syncer.ignore-table]]
db-name = "test"
tbl-name = "log"

[[syncer.ignore-table]]
db-name = "test"
tbl-name = "audit"
```

If all changes in a binlog file are filtered, the whole binlog file is ignored.

Default value: []

replicate-do-db

- Specifies the database to be replicated. For example, [db1, db2].
- Default value: []

replicate-do-table

Specifies the table to be replicated. For example:

```
[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "log"

[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "~^a.*"
```

Default value: []

txn-batch

- When the downstream is the `mysql` or `tidb` type, DML operations are executed in different batches. This parameter specifies how many DML operations can be included in each transaction.
- Default value: 20

worker-count

- When the downstream is the `mysql` or `tidb` type, DML operations are executed concurrently. This parameter specifies the concurrency numbers of DML operations.
- Default value: 16

disable-dispatch

- Disables the concurrency and forcibly set `worker-count` to 1.
- Default value: `false`

safe-mode

If the safe mode is enabled, Drainer modifies the replication updates in the following way:

- `Insert` is modified to `Replace Into`
- `Update` is modified to `Delete` plus `Replace Into`

Default value: `false`

#### 10.7.5.2.14 syncer.to

The `syncer.to` section introduces different types of downstream configuration items according to configuration types.

mysql/tidb

The following configuration items are related to connection to downstream databases:

- `host`: If this item is not set, TiDB Binlog tries to check the `MYSQL_HOST` environment variable which is `localhost` by default.
- `port`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PORT` environment variable which is `3306` by default.
- `user`: If this item is not set, TiDB Binlog tries to check the `MYSQL_USER` environment variable which is `root` by default.
- `password`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PSWD` environment variable which is `" "` by default.

file

- `dir`: Specifies the directory where binlog files are stored. If this item is not set, `data-dir` is used.

kafka

When the downstream is Kafka, the valid configuration items are as follows:

- `zookeeper-addrs`
- `kafka-addrs`
- `kafka-version`
- `kafka-max-messages`
- `kafka-max-message-size`
- `topic-name`

#### 10.7.5.2.15 syncer.to.checkpoint

- `type`: Specifies in what way the replication progress is saved. Currently, the available options are `mysql`, `tidb`, and `file`.

This configuration item is the same as the downstream type by default. For example, when the downstream is `file`, the checkpoint progress is saved in the local file `<data-dir>/savepoint`; when the downstream is `mysql`, the progress is saved in the downstream database. If you need to explicitly specify using `mysql` or `tidb` to store the progress, make the following configuration:

- `schema`: `"tidb_binlog"` by default.

**Note:**

When deploying multiple Drainer nodes in the same TiDB cluster, you need to specify a different checkpoint schema for each node. Otherwise, the replication progress of two instances will overwrite each other.

- host
- user
- password
- port

#### 10.7.5.3 TiDB Binlog Configuration File

This document introduces the configuration items of TiDB Binlog.

##### 10.7.5.3.1 Pump

This section introduces the configuration items of Pump. For the example of a complete Pump configuration file, see [Pump Configuration](#).

addr

- Specifies the listening address of HTTP API in the format of host:port.
- Default value: 127.0.0.1:8250

advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of host:port.
- Default value: 127.0.0.1:8250

socket

- The Unix socket address that HTTP API listens to.
- Default value: “”

pd-urls

- Specifies the comma-separated list of PD URLs. If multiple addresses are specified, when the PD client fails to connect to one address, it automatically tries to connect to another address.

- Default value: `http://127.0.0.1:2379`

data-dir

- Specifies the directory where binlogs and their indexes are stored locally.
- Default value: `data.pump`

heartbeat-interval

- Specifies the heartbeat interval (in seconds) at which the latest status is reported to PD.
- Default value: 2

gen-binlog-interval

- Specifies the interval (in seconds) at which data is written into fake binlog.
- Default value: 3

gc

- Specifies the number of days (integer) that binlogs can be stored locally. Binlogs stored longer than the specified number of days are automatically deleted.
- Default value: 7

log-file

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: “”

log-level

- Specifies the log level.
- Default value: `info`

node-id

- Specifies the Pump node ID. With this ID, this Pump process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8250`.

## security

This section introduces configuration items related to security.

### ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, `/path/to/ca.pem`.
- Default value: `""`

### ssl-cert

- Specifies the path of the X509 certificate file encoded in the Privacy Enhanced Mail (PEM) format. For example, `/path/to/pump.pem`.
- Default value: `""`

### ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, `/path/to/pump-key.pem`.
- Default value: `""`

## storage

This section introduces configuration items related to storage.

### sync-log

- Specifies whether to use `fsync` after each **batch** write to binlog to ensure data safety.
- Default value: `true`

### kv\_chan\_cap

- Specifies the number of write requests that the buffer can store before Pump receives these requests.
- Default value: `1048576` (that is, 2 to the power of 20)

### slow\_write\_threshold

- The threshold (in seconds). If it takes longer to write a single binlog file than this specified threshold, the write is considered slow write and "take a long time to `→ write binlog`" is output in the log.
- Default value: `1`

stop-write-at-available-space

- Binlog write requests are no longer accepted when the available storage space is below this specified value. You can use the format such as 900 MB, 5 GB, and 12 GiB to specify the storage space. If there is more than one Pump node in the cluster, when a Pump node refuses a write request because of the insufficient space, TiDB will automatically write binlogs to other Pump nodes.
- Default value: 10 GiB

kv

Currently the storage of Pump is implemented based on [GoLevelDB](#). Under `storage` there is also a `kv` subgroup that is used to adjust the GoLevel configuration. The supported configuration items are shown as below:

- `block-cache-capacity`
- `block-restart-interval`
- `block-size`
- `compaction-L0-trigger`
- `compaction-table-size`
- `compaction-total-size`
- `compaction-total-size-multiplier`
- `write-buffer`
- `write-L0-pause-trigger`
- `write-L0-slowdown-trigger`

For the detailed description of the above items, see [GoLevelDB Document](#).

#### 10.7.5.3.2 Drainer

This section introduces the configuration items of Drainer. For the example of a complete Drainer configuration file, see [Drainer Configuration](#)

addr

- Specifies the listening address of HTTP API in the format of `host:port`.
- Default value: 127.0.0.1:8249

advertise-addr

- Specifies the externally accessible HTTP API address. This address is registered in PD in the format of `host:port`.
- Default value: 127.0.0.1:8249

log-file

- Specifies the path where log files are stored. If the parameter is set to an empty value, log files are not stored.
- Default value: “”

log-level

- Specifies the log level.
- Default value: `info`

node-id

- Specifies the Drainer node ID. With this ID, this Drainer process can be identified in the cluster.
- Default value: `hostname:port number`. For example, `node-1:8249`.

data-dir

- Specifies the directory used to store files that need to be saved during Drainer operation.
- Default value: `data.drainer`

detect-interval

- Specifies the interval (in seconds) at which PD updates the Pump information.
- Default value: 5

pd-urls

- The comma-separated list of PD URLs. If multiple addresses are specified, the PD client will automatically attempt to connect to another address if an error occurs when connecting to one address.
- Default value: `http://127.0.0.1:2379`

initial-commit-ts

- Specifies from which commit timestamp of the transaction the replication process starts. This configuration is applicable only to the Drainer node that is in the replication process for the first time. If a checkpoint already exists in the downstream, the replication will be performed according to the time recorded in the checkpoint.
- `commit ts` (`commit timestamp`) is a specific point in time for `transaction` commits in TiDB. It is a globally unique and increasing timestamp from PD as the unique ID of the current transaction. You can get the `initial-commit-ts` configuration in the following typical ways:

- If BR is used, you can get `initial-commit-ts` from the backup TS recorded in the metadata backed up by BR (`backupmeta`).
  - If Dumpling is used, you can get `initial-commit-ts` from the Pos recorded in the metadata backed up by Dumpling (`metadata`),
  - If PD Control is used, `initial-commit-ts` is in the output of the `tso` command.
- Default value: `-1`. Drainer will get a new timestamp from PD as the starting time, which means that the replication process starts from the current time.

synced-check-time

- You can access the `/status` path through the HTTP API to query the status of Drainer replication. `synced-check-time` specifies how many minutes from the last successful replication is considered as `synced`, that is, the replication is complete.
- Default value: `5`

compressor

- Specifies the compression algorithm used for data transfer between Pump and Drainer. Currently only the `gzip` algorithm is supported.
- Default value: `""`, which means no compression.

security

This section introduces configuration items related to security.

ssl-ca

- Specifies the file path of the trusted SSL certificate list or CA list. For example, `/path/to/ca.pem`.
- Default value: `""`

ssl-cert

- Specifies the path of the X509 certificate file encoded in the PEM format. For example, `/path/to/drainer.pem`.
- Default value: `""`

ssl-key

- Specifies the path of the X509 key file encoded in the PEM format. For example, `/path/to/pump-key.pem`.
- Default value: `""`

## syncer

The `syncer` section includes configuration items related to the downstream.

### db-type

Currently, the following downstream types are supported:

- `mysql`
- `tidb`
- `kafka`
- `file`

Default value: `mysql`

### sql-mode

- Specifies the SQL mode when the downstream is the `mysql` or `tidb` type. If there is more than one mode, use commas to separate them.
- Default value: “”

### ignore-txn-commit-ts

- Specifies the commit timestamp at which the binlog is ignored, such as `[416815754209656834, ↪ 421349811963822081]`.
- Default value: `[]`

### ignore-schemas

- Specifies the database to be ignored during replication. If there is more than one database to be ignored, use commas to separate them. If all changes in a binlog file are filtered, the whole binlog file is ignored.
- Default value: `INFORMATION_SCHEMA, PERFORMANCE_SCHEMA, mysql`

### ignore-table

Ignores the specified table changes during replication. You can specify multiple tables to be ignored in the `toml` file. For example:

```
[[syncer.ignore-table]]
db-name = "test"
tbl-name = "log"

[[syncer.ignore-table]]
db-name = "test"
tbl-name = "audit"
```

If all changes in a binlog file are filtered, the whole binlog file is ignored.

Default value: []

replicate-do-db

- Specifies the database to be replicated. For example, [db1, db2].
- Default value: []

replicate-do-table

Specifies the table to be replicated. For example:

```
[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "log"

[[syncer.replicate-do-table]]
db-name ="test"
tbl-name = "~^a.*"
```

Default value: []

txn-batch

- When the downstream is the mysql or tidb type, DML operations are executed in different batches. This parameter specifies how many DML operations can be included in each transaction.
- Default value: 20

worker-count

- When the downstream is the mysql or tidb type, DML operations are executed concurrently. This parameter specifies the concurrency numbers of DML operations.
- Default value: 16

disable-dispatch

- Disables the concurrency and forcibly set worker-count to 1.
- Default value: false

safe-mode

If the safe mode is enabled, Drainer modifies the replication updates in the following way:

- `Insert` is modified to `Replace Into`
- `Update` is modified to `Delete` plus `Replace Into`

Default value: `false`

`syncer.to`

The `syncer.to` section introduces different types of downstream configuration items according to configuration types.

`mysql/tidb`

The following configuration items are related to connection to downstream databases:

- `host`: If this item is not set, TiDB Binlog tries to check the `MYSQL_HOST` environment variable which is `localhost` by default.
- `port`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PORT` environment variable which is `3306` by default.
- `user`: If this item is not set, TiDB Binlog tries to check the `MYSQL_USER` environment variable which is `root` by default.
- `password`: If this item is not set, TiDB Binlog tries to check the `MYSQL_PSWD` environment variable which is `" "` by default.

`file`

- `dir`: Specifies the directory where binlog files are stored. If this item is not set, `data-dir` is used.

`kafka`

When the downstream is Kafka, the valid configuration items are as follows:

- `zookeeper-addrs`
- `kafka-addrs`
- `kafka-version`
- `kafka-max-messages`
- `kafka-max-message-size`
- `topic-name`

`syncer.to.checkpoint`

- `type`: Specifies in what way the replication progress is saved. Currently, the available options are `mysql`, `tidb`, and `file`.

This configuration item is the same as the downstream type by default. For example, when the downstream is `file`, the checkpoint progress is saved in the local file `<data-dir>/savepoint`; when the downstream is `mysql`, the progress is saved in the downstream database. If you need to explicitly specify using `mysql` or `tidb` to store the progress, make the following configuration:

- schema: "tidb\_binlog" by default.

**Note:**

When deploying multiple Drainer nodes in the same TiDB cluster, you need to specify a different checkpoint schema for each node. Otherwise, the replication progress of two instances will overwrite each other.

- host
- user
- password
- port

## 10.7.6 Upgrade TiDB Binlog

This document introduces how to upgrade TiDB Binlog that is deployed manually to the latest `cluster` version. There is also a section on how to upgrade TiDB Binlog from an earlier incompatible version (Kafka/Local version) to the latest version.

### 10.7.6.1 Upgrade TiDB Binlog deployed manually

Follow the steps in this section if you deploy TiDB Binlog manually.

#### 10.7.6.1.1 Upgrade Pump

First, upgrade each Pump instance in the cluster one by one. This ensures that there are always Pump instances in the cluster that can receive binlogs from TiDB. The steps are as below:

1. Replace the original file with the new version of `pump`.
2. Restart the Pump process.

#### 10.7.6.1.2 Upgrade Drainer

Second, upgrade the Drainer component:

1. Replace the original file with the new version of `drainer`.
2. Restart the Drainer process.

### 10.7.6.2 Upgrade TiDB Binlog from Kafka/Local version to the cluster version

The new TiDB versions (v2.0.8-binlog, v2.1.0-rc.5 or later) are not compatible with the Kafka version or Local version of TiDB Binlog. If TiDB is upgraded to one of the new versions, it is required to use the cluster version of TiDB Binlog. If the Kafka or local version of TiDB Binlog is used before upgrading, you need to upgrade your TiDB Binlog to the cluster version.

The corresponding relationship between TiDB Binlog versions and TiDB versions is shown in the following table:

TiDB Binlog version	TiDB version	Note
Local	TiDB 1.0 or earlier	
Kafka	TiDB 1.0 ~ TiDB 2.1 RC5	TiDB 1.0 supports both the local and Kafka versions of TiDB Binlog.
Cluster	TiDB v2.0.8-binlog, TiDB 2.1 RC5 or later	TiDB v2.0.8-binlog is a special 2.0 version supporting the cluster version of TiDB Binlog.

#### 10.7.6.2.1 Upgrade process

##### Note:

If importing the full data is acceptable, you can abandon the old version and deploy TiDB Binlog following [TiDB Binlog Cluster Deployment](#).

If you want to resume replication from the original checkpoint, perform the following steps to upgrade TiDB Binlog:

1. Deploy the new version of Pump.
  2. Stop the TiDB cluster service.
  3. Upgrade TiDB and the configuration, and write the binlog data to the new Pump cluster.
  4. Reconnect the TiDB cluster to the service.
  5. Make sure that the old version of Drainer has replicated the data in the old version of Pump to the downstream completely;
- Query the `status` interface of Drainer, command as below:

```
curl 'http://172.16.10.49:8249/status'
```

```
{"PumpPos": {"172.16.10.49:8250": {"offset": 32686}}, "Synced": true, "  
    ↪ DepositWindow": {"Upper": 398907800202772481, "Lower"  
    ↪ ": 398907799455662081}"}
```

If the return value of `Synced` is True, it means Drainer has replicated the data in the old version of Pump to the downstream completely.

6. Start the new version of Drainer.
7. Close the Pump and Drainer of the old versions and the dependent Kafka and ZooKeeper.

## 10.7.7 TiDB Binlog Monitoring

After you have deployed TiDB Binlog successfully, you can go to the Grafana Web (default address: [http://grafana\\_ip:3000](http://grafana_ip:3000), default account: admin, password: admin) to check the state of Pump and Drainer.

### 10.7.7.1 Monitoring metrics

TiDB Binlog consists of two components: Pump and Drainer. This section shows the monitoring metrics of Pump and Drainer.

#### 10.7.7.1.1 Pump monitoring metrics

To understand the Pump monitoring metrics, check the following table:

Pump mon- itor- ing met- rics	Description
Storage	Records Size the to- tal disk space (ca- pac- ity) and the avail- able disk space (avail- able)

---

Pump mon- itor- ing met- rics	Description
MetadataRecords	the biggest TSO (gc_tso ↳ ) of the bin- log that each Pump node can delete, and the biggest com- mit TSO (max_commit_tso ↳ ) of the saved bin- log

---

Pump mon- itor- ing met- rics	Description
Write Bin- log QPS by In- stance	Shows QPS of writ- ing bin- log
re- quests re- ceived by each	re-
Pump node	quests
Write Bin- log La- tency	re-
tency time of each	ceived
Pump node	by
writ- ing	each
bin- log	time

---

Pump	mon-	itor-	ing	met-	rics	Description
Storage	Shows					
Write	the					
Bin-	size					
log	of					
Size	the					
	bin-					
	log					
	data					
	writ-					
	ten					
	by					
	Pump					
Storage	Records					
Write	the					
Bin-	la-					
log	tency					
La-	time					
tency	of					
	the					
	Pump					
	stor-					
	age					
	mod-					
	ule					
	writ-					
	ing					
	bin-					
	log					

---

Pump mon- itor- ing met- rics	Description
Pump	Records
Stor-	the
age	num-
Er-	ber
ror	of
By	er-
Type	rors
	en-
	coun-
	tered
	by
	Pump,
	counted
	based
	on
	the
	type
	of
	er-
	ror
Query	The
TiKV	num-
	ber
	of
	times
	that
	Pump
	queries
	the
	trans-
	ac-
	tion
	sta-
	tus
	through
	TiKV

---

#### 10.7.7.1.2 Drainer monitoring metrics

To understand the Drainer monitoring metrics, check the following table:



---

Drainer	mon-	itor-	ing	met-	rics	Description
---------	------	-------	-----	------	------	-------------

---

Drainer	mon-	itor-	ing	met-	rics	Description
---------	------	-------	-----	------	------	-------------

---

Checkpoints	Shows
-------------	-------

TSO	the
	bigest
TSO	
	time
	of
	the
	bin-
	log
	that
Drainer	
has	
al-	
ready	
repli-	
cated	
into	
the	
down-	
stream.	

You	
can	
get	
the	
lag	
by	
us-	
ing	
the	
cur-	
rent	
time	
to	

1024 sub-

tract  
the  
bin-

---

Drainer mon- itor- ing met- rics	Description
Pump	Records
Han-	the
dle	biggest
TSO	TSO
	time
	among
	the
	bin-
	log
	files
	that
	Drainer
	ob-
	tains
	from
	each
Pump	
node	
Pull	Shows
Bin-	the
log	QPS
QPS	when
by	Drainer
Pump	ob-
NodeID	ains
	bin-
	log
	from
	each
Pump	
node	

---

rics	Description
95%	Records
Bin-	the
log	de-
Reach	lay
Du-	from
ra-	the
tion	time
By	when
Pump	bin-
	log
	is
	writ-
	ten
	into
Pump	
	to
	the
	time
	when
	the
	bin-
	log
	is
	ob-
	tained
	by
	Drainer

---

Drainer mon- itor- ing met- rics	Description
Error	Shows By the Type num- ber of er- rors en- coun- tered by Drainer, counted based on the type of er- ror
SQL	Records Query the Time time it takes Drainer to exe- cute the SQL state- ment in the down- stream

Drainer mon- itor- ing met- rics	Description
Draine	Shows Event the num- ber of vari- ous types of events, in- clud- ing “ddl”, “in- sert”, “delete”, “up- date”, “flush”, and “save- point”

Drainer mon- itor- ing met- rics	Description
ExecuteRecords	
Time	the time it takes to write bin- log into the down- stream sync- ing mod- ule
95%	Shows Bin- log Size
	the size of the bin- log data that
	Drainer ob- tains from each Pump node

---

rics	Description
DDL	Records
Job	the
Count	num- ber of
	DDL
	state- ments
	han- dled
	by
	Drainer
Queue	Records
Size	the work queue size in
	Drainer

---

### 10.7.7.2 Alert rules

This section gives the alert rules for TiDB Binlog. According to the severity level, TiDB Binlog alert rules are divided into three categories (from high to low): emergency-level, critical-level and warning-level.

#### 10.7.7.2.1 Emergency-level alerts

Emergency-level alerts are often caused by a service or node failure. Manual intervention is required immediately.

`binlog_pump_storage_error_count`

- Alert rule:

```
changes(binlog_pump_storage_error_count[1m]) > 0
```

- Description:

Pump fails to write the binlog data to the local storage.

- Solution:

Check whether an error exists in the `pump_storage_error` monitoring and check the Pump log to find the causes.

#### 10.7.7.2.2 Critical-level alerts

For the critical-level alerts, a close watch on the abnormal metrics is required.

`binlog_drainer_checkpoint_high_delay`

- Alert rule:

```
(time() - binlog_drainer_checkpoint_tso / 1000) > 3600
```

- Description:

The delay of Drainer replication exceeds one hour.

- Solution:

- Check whether it is too slow to obtain the data from Pump:

You can check `handle tso` of Pump to get the time for the latest message of each Pump. Check whether a high latency exists for Pump and make sure the corresponding Pump is running normally.

- Check whether it is too slow to replicate data in the downstream based on Drainer `event` and Drainer `execute latency`:

- \* If Drainer `execute time` is too large, check the network bandwidth and latency between the machine with Drainer deployed and the machine with the target database deployed, and the state of the target database.
- \* If Drainer `execute time` is not too large and Drainer `event` is too small, add `work count` and `batch` and retry.

- If the two solutions above cannot work, contact [support@pingcap.com](mailto:support@pingcap.com).

#### 10.7.7.2.3 Warning-level alerts

Warning-level alerts are a reminder for an issue or error.

`binlog_pump_write_binlog_rpc_duration_seconds_bucket`

- Alert rule:

```
histogram_quantile(0.9, rate(binlog_pump_rpc_duration_seconds_bucket{  
    ↳ method="WriteBinlog"}[5m])) > 1
```

- Description:

It takes too much time for Pump to handle the TiDB request of writing binlog.

- Solution:

- Verify the disk performance pressure and check the disk performance monitoring via `node exported`.
- If both `disk latency` and `util` are low, contact [support@pingcap.com](mailto:support@pingcap.com).

`binlog_pump_storage_write_binlog_duration_time_bucket`

- Alert rule:

```
histogram_quantile(0.9, rate(binlog_pump_storage_write_binlog_duration_time_bucket
    ↪ {type="batch"})[5m])) > 1
```

- Description:

The time it takes for Pump to write the local binlog to the local disk.

- Solution:

Check the state of the local disk of Pump and fix the problem.

`binlog_pump_storage_available_size_less_than_20G`

- Alert rule:

```
binlog_pump_storage_storage_size_bytes{type="available"} < 20 * 1024 *
    ↪ 1024 * 1024
```

- Description:

The available disk space of Pump is less than 20 GB.

- Solution:

Check whether Pump `gc_tso` is normal. If not, adjust the GC time configuration of Pump or get the corresponding Pump offline.

`binlog_drainer_checkpoint_tso_no_change_for_1m`

- Alert rule:

```
changes(binlog_drainer_checkpoint_tso[1m]) < 1
```

- Description:

Drainer `checkpoint` has not been updated for one minute.

- Solution:

Check whether all the Pumps that are not offline are running normally.

`binlog_drainer_execute_duration_time_more_than_10s`

- Alert rule:

```
histogram_quantile(0.9, rate(binlog_drainer_execute_duration_time_bucket
    ↪ [1m])) > 10
```

- Description:

The transaction time it takes Drainer to replicate data to TiDB. If it is too large, the Drainer replication of data is affected.

- Solution:

- Check the TiDB cluster state.
- Check the Drainer log or monitor. If a DDL operation causes this problem, you can ignore it.

## 10.7.8 Reparo User Guide

Reparo is a TiDB Binlog tool, used to recover the incremental data. To back up the incremental data, you can use Drainer of TiDB Binlog to output the binlog data in the protobuf format to files. To restore the incremental data, you can use Reparo to parse the binlog data in the files and apply the binlog in TiDB/MySQL.

Download Reparo via [tidb-binlog-cluster-latest-linux-amd64.tar.gz](#)

### 10.7.8.1 Reparo usage

#### 10.7.8.1.1 Description of command line parameters

Usage of Reparo:

```
-L string
    The level of the output information of logs
    Value: "debug"/"info"/"warn"/"error"/"fatal" ("info" by default)
-V Prints the version.
-c int
    The number of concurrencies in the downstream for the replication
    ↪ process (`16` by default). A higher value indicates a better
    ↪ throughput for the replication.
-config string
    The path of the configuration file
    If the configuration file is specified, Reparo reads the configuration
    ↪ data in this file.
    If the configuration data also exists in the command line parameters,
    ↪ Reparo uses the configuration data in the command line parameters
    ↪ to cover that in the configuration file.
-data-dir string
```

```

The storage directory for the binlog file in the protobuf format that
    ↪ Drainer outputs ("data.drainer" by default)
-dest-type string
    The downstream service type
    Value: "print"/"mysql" ("print" by default)
    If it is set to "print", the data is parsed and printed to standard
        ↪ output while the SQL statement is not executed.
    If it is set to "mysql", you need to configure the "host", "port", "user"
        ↪ " and "password" information in the configuration file.
-log-file string
    The path of the log file
-log-rotate string
    The switch frequency of log files
    Value: "hour"/"day"
-start-datetime string
    Specifies the time point for starting recovery.
    Format: "2006-01-02 15:04:05"
    If it is not set, the recovery process starts from the earliest binlog
        ↪ file.
-stop-datetime string
    Specifies the time point of finishing the recovery process.
    Format: "2006-01-02 15:04:05"
    If it is not set, the recovery process ends up with the last binlog file
        ↪ .
-safe-mode bool
    Specifies whether to enable safe mode. When enabled, it supports
        ↪ repeated replication.
-txn-batch int
    The number of SQL statements in a transaction that is output to the
        ↪ downstream database (`20` by default).

```

#### 10.7.8.1.2 Description of the configuration file

```

### The storage directory for the binlog file in the protobuf format that
    ↪ Drainer outputs
data-dir = "./data.drainer"

### The level of the output information of logs
### Value: "debug"/"info"/"warn"/"error"/"fatal" ("info" by default)
log-level = "info"

### Uses `start-datetime` and `stop-datetime` to specify the time range in
    ↪ which
### the binlog files are to be recovered.

```

```

### Format: "2006-01-02 15:04:05"
### start-datetime = ""
### stop-datetime = ""

### Correspond to `start-datetime` and `stop-datetime` respectively.
### They are used to specify the time range in which the binlog files are to
    ↪ be recovered.
### If `start-datetime` and `stop-datetime` are set, there is no need to set
    ↪ `start-tso` and `stop-tso`.
### start-tso = 0
### stop-tso = 0

### The downstream service type
### Value: "print"/"mysql" ("print" by default)
### If it is set to "print", the data is parsed and printed to standard
    ↪ output
### while the SQL statement is not executed.
### If it is set to "mysql", you need to configure `host`, `port`, `user`
    ↪ and `password` in [dest-db].
dest-type = "mysql"

### The number of SQL statements in a transaction that is output to the
    ↪ downstream database (`20` by default).
txn-batch = 20

### The number of concurrencies in the downstream for the replication
    ↪ process (`16` by default). A higher value indicates a better
    ↪ throughput for the replication.
worker-count = 16

### Safe-mode configuration
### Value: "true"/"false" ("false" by default)
### If it is set to "true", Reparo splits the `UPDATE` statement into a `
    ↪ DELETE` statement plus a `REPLACE` statement.
safe-mode = false

### `replicate-do-db` and `replicate-do-table` specify the database and
    ↪ table to be recovered.
### `replicate-do-db` has priority over `replicate-do-table`.
### You can use a regular expression for configuration. The regular
    ↪ expression should start with "~".
### The configuration method for `replicate-do-db` and `replicate-do-table`
    ↪ is
### the same with that for `replicate-do-db` and `replicate-do-table` of
    ↪ Drainer.

```

```

### replicate-do-db = ["~^b.*", "s1"]
### [[replicate-do-table]]
### db-name ="test"
### tbl-name = "log"
### [[replicate-do-table]]
### db-name ="test"
### tbl-name = "~^a.*"

### If `dest-type` is set to `mysql`, `dest-db` needs to be configured.
[dest-db]
host = "127.0.0.1"
port = 3309
user = "root"
password =

```

#### 10.7.8.1.3 Start example

```
./bin/reparo -config reparo.toml
```

**Note:**

- **data-dir** specifies the directory for the binlog file that Drainer outputs.
- Both **start-datetime** and **start-tso** are used to specify the time point for starting recovery, but they are different in the time format. If they are not set, the recovery process starts from the earliest binlog file by default.
- Both **stop-datetime** and **stop-tso** are used to specify the time point for finishing recovery, but they are different in the time format. If they are not set, the recovery process ends up with the last binlog file by default.
- **dest-type** specifies the destination type. Its value can be “mysql” and “print.”
  - When it is set to **mysql**, the data can be recovered to MySQL or TiDB that uses or is compatible with the MySQL protocol. In this case, you need to specify the database information in **[dest-db]** of the configuration information.
  - When it is set to **print**, only the binlog information is printed. It is generally used for debugging and checking the binlog information. In this case, there is no need to specify **[dest-db]**.

- `replicate-do-db` specifies the database for recovery. If it is not set, all the databases are to be recovered.
- `replicate-do-table` specifies the table for recovery. If it is not set, all the tables are to be recovered.

### 10.7.9 binlogctl

Binlog Control (`binlogctl` for short) is a command line tool for TiDB Binlog. You can use `binlogctl` to manage TiDB Binlog clusters.

You can use `binlogctl` to:

- Check the state of Pump or Drainer
- Pause or close Pump or Drainer
- Handle the abnormal state of Pump or Drainer

The following are its usage scenarios:

- An error occurs during data replication or you need to check the running state of Pump or Drainer.
- You need to pause or close Pump or Drainer when maintaining the cluster.
- A Pump or Drainer process exits abnormally, while the node state is not updated or is unexpected. This affects the data replication task.

#### 10.7.9.1 Download `binlogctl`

##### Note:

It is recommended that the version of the Control tool you use is consistent with the version of the cluster.

```
wget https://download.pingcap.org/tidb-{version}-linux-amd64.tar.gz &&
wget https://download.pingcap.org/tidb-{version}-linux-amd64.sha256
```

To check the file integrity, execute the following command. If the result is OK, the file is correct.

```
sha256sum -c tidb-{version}-linux-amd64.sha256
```

To check the file integrity, execute the following command. If the result is OK, the file is correct.

```
sha256sum -c tidb-enterprise-tools-latest-linux-amd64.sha256
```

### 10.7.9.2 Descriptions

Command line parameters:

```
Usage of binlogctl:
  -V      prints version and exit
  -cmd string
    operator: "generate_meta", "pumps", "drainers", "update-pump", "
      ↪ update-drainer", "pause-pump", "pause-drainer", "offline-pump
      ↪ ", "offline-drainer", "encrypt" (default "pumps")
  -data-dir string
    meta directory path (default "binlog_position")
  -node-id string
    id of node, used to update some nodes with operations update-pump,
    ↪ update-drainer, pause-pump, pause-drainer, offline-pump and
    ↪ offline-drainer
  -pd-urls string
    a comma separated list of PD endpoints (default "http
      ↪ ://127.0.0.1:2379")
  -show-offline-nodes
    include offline nodes when querying pumps/drainers
  -ssl-ca string
    Path of file that contains list of trusted SSL CAs for connection
    ↪ with cluster components.
  -ssl-cert string
    Path of file that contains X509 certificate in PEM format for
    ↪ connection with cluster components.
  -ssl-key string
    Path of file that contains X509 key in PEM format for connection with
    ↪ cluster components.
  -state string
    set node's state, can be set to online, pausing, paused, closing or
    ↪ offline.
  -text string
    text to be encrypted when using encrypt command
  -time-zone Asia/Shanghai
    set time zone if you want to save time info in savepoint file; for
    ↪ example, Asia/Shanghai for CST time, `Local` for local time
```

Command examples:

- Check the state of all the Pump or Drainer nodes.

Set cmd to pumps or drainers. For example:

```
bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd pumps
```

```
[2019/04/28 09:29:59.016 +00:00] [INFO] [nodes.go:48] ["query node"] [
    ↳ type=pump] [node="{NodeID: 1.1.1.1:8250, Addr: pump:8250, State:
    ↳ online, MaxCommitTS: 408012403141509121, UpdateTime: 2019-04-28
    ↳ 09:29:57 +0000 UTC}"]
```

```
bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd drainers
```

```
[2019/04/28 09:29:59.016 +00:00] [INFO] [nodes.go:48] ["query node"] [
    ↳ type=drainer] [node="{NodeID: 1.1.1.1:8249, Addr: 1.1.1.1:8249,
    ↳ State: online, MaxCommitTS: 408012403141509121, UpdateTime:
    ↳ 2019-04-28 09:29:57 +0000 UTC}"]
```

- Pause or close Pump or Drainer.

You can use the following commands to pause or close services:

Command	Description	Example
pause-pump	Pause Pump	bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd pause-pump -node-id ip-127-0-0-1:8250
pause-drainer	Pause Drainer	bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd pause-drainer -node-id ip-127-0-0-1:8249
offline-pump	Close Pump	bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd offline-pump -node-id ip-127-0-0-1:8250
offline-drainer	Close Drainer	bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd offline-drainer -node-id ip-127-0-0-1:8249

binlogctl sends the HTTP request to the Pump or Drainer node. After receiving the request, the node executes the exiting procedures accordingly.

- Modify the state of a Pump or Drainer node in abnormal states.

When a Pump or Drainer node runs normally or when it is paused or closed in the normal process, it is in the normal state. In abnormal states, the Pump or Drainer node cannot correctly maintain its state. This affects data replication tasks. In this case, use binlogctl to repair the state information.

To update the state of a Pump or Drainer node, set cmd to update-pump or update-drainer. The state can be paused or offline. For example:

```
bin/binlogctl -pd-urls=http://127.0.0.1:2379 -cmd update-pump -node-id
    ↳ ip-127-0-0-1:8250 -state paused
```

### Note:

When a Pump or Drainer node runs normally, it regularly updates its state to PD. The above command directly modifies the Pump or Drainer state saved in PD; therefore, do not use the command when the Pump or Drainer node runs normally. For more information, refer to [TiDB Binlog FAQ](#).

## 10.7.10 Binlog Consumer Client User Guide

Binlog Consumer Client is used to consume TiDB secondary binlog data from Kafka and output the data in a specific format. Currently, Drainer supports multiple kinds of downstream, including MySQL, TiDB, file and Kafka. But sometimes users have customized requirements for outputting data to other formats, for example, Elasticsearch and Hive, so this feature is introduced.

### 10.7.10.1 Configure Drainer

Modify the configuration file of Drainer and set it to output the data to Kafka:

```
[syncer]
db-type = "kafka"

[syncer.to]
### the Kafka address
kafka-addrs = "127.0.0.1:9092"
### the Kafka version
kafka-version = "0.8.2.0"
```

### 10.7.10.2 Customized development

#### 10.7.10.2.1 Data format

Firstly, you need to obtain the format information of the data which is output to Kafka by Drainer:

```
// `Column` stores the column data in the corresponding variable based on
// the data type.
message Column {
    // Indicates whether the data is null
    optional bool is_null = 1 [ default = false ];
    // Stores `int` data
    optional int64 int64_value = 2;
    // Stores `uint`, `enum`, and `set` data
```

```

optional uint64 uint64_value = 3;
// Stores `float` and `double` data
optional double double_value = 4;
// Stores `bit`, `blob`, `binary` and `json` data
optional bytes bytes_value = 5;
// Stores `date`, `time`, `decimal`, `text`, `char` data
optional string string_value = 6;
}

// `ColumnInfo` stores the column information, including the column name,
// type, and whether it is the primary key.
message ColumnInfo {
    optional string name = 1 [ (gogoproto.nullable) = false ];
    // the lower case column field type in MySQL
    // https://dev.mysql.com/doc/refman/8.0/en/data-types.html
    // for the `numeric` type: int bigint smallint tinyint float double
    // decimal bit
    // for the `string` type: text longtext mediumtext char tinytext varchar
    // blob longblob mediumblob binary tinyblob varbinary
    // enum set
    // for the `json` type: json
    optional string mysql_type = 2 [ (gogoproto.nullable) = false ];
    optional bool is_primary_key = 3 [ (gogoproto.nullable) = false ];
}

// `Row` stores the actual data of a row.
message Row { repeated Column columns = 1; }

// `MutationType` indicates the DML type.
enum MutationType {
    Insert = 0;
    Update = 1;
    Delete = 2;
}

// `Table` contains mutations in a table.
message Table {
    optional string schema_name = 1;
    optional string table_name = 2;
    repeated ColumnInfo column_info = 3;
    repeated TableMutation mutations = 4;
}

// `TableMutation` stores mutations of a row.
message TableMutation {
}

```

```

required MutationType type = 1;
// data after modification
required Row row = 2;
// data before modification. It only takes effect for `Update MutationType
// → `.
optional Row change_row = 3;
}

// `DMLData` stores all the mutations caused by DML in a transaction.
message DMLData {
    // `tables` contains all the table changes in the transaction.
    repeated Table tables = 1;
}

// `DDLData` stores the DDL information.
message DDLData {
    // the database used currently
    optional string schema_name = 1;
    // the relates table
    optional string table_name = 2;
    // `ddl_query` is the original DDL statement query.
    optional bytes ddl_query = 3;
}

// `BinlogType` indicates the binlog type, including DML and DDL.
enum BinlogType {
    DML = 0; // Has `dml_data`
    DDL = 1; // Has `ddl_query`
}

// `Binlog` stores all the changes in a transaction. Kafka stores the
// → serialized result of the structure data.
message Binlog {
    optional BinlogType type = 1 [ (gogoproto.nullable) = false ];
    optional int64 commit_ts = 2 [ (gogoproto.nullable) = false ];
    optional DMLData dml_data = 3;
    optional DDLData ddl_data = 4;
}

```

For the definition of the data format, see [secondary\\_binlog.proto](#)

#### 10.7.10.2.2 Driver

The [TiDB-Tools](#) project provides [Driver](#), which is used to read the binlog data in Kafka. It has the following features:

- Read the Kafka data.
- Locate the binlog stored in Kafka based on `commit ts`.

You need to configure the following information when using Driver:

- `KafkaAddr`: the address of the Kafka cluster
- `CommitTS`: from which `commit ts` to start reading the binlog
- `Offset`: from which Kafka `offset` to start reading data. If `CommitTS` is set, you needn't configure this parameter.
- `ClusterID`: the cluster ID of the TiDB cluster
- `Topic`: the topic name of Kafka. If `Topic` is empty, use the default name in Drainer `<ClusterID>_obinlog`.

You can use Driver by quoting the Driver code in package and refer to the example code provided by Driver to learn how to use Driver and parse the binlog data.

Currently, two examples are provided:

- Using Driver to replicate data to MySQL. This example shows how to convert a binlog to SQL
- Using Driver to print data

#### Note:

- The example code only shows how to use Driver. If you want to use Driver in the production environment, you need to optimize the code.
- Currently, only the Golang version of Driver and example code are available. If you want to use other languages, you need to generate the code file in the corresponding language based on the binlog proto file and develop an application to read the binlog data in Kafka, parse the data, and output the data to the downstream. You are also welcome to optimize the example code and submit the example code of other languages to [TiDB-Tools](#).

### 10.7.11 TiDB Binlog Relay Log

When replicating binlogs, Drainer splits transactions from the upstream and replicates the split transactions concurrently to the downstream.

In extreme cases where the upstream clusters are not available and Drainer exits abnormally, the downstream clusters (MySQL or TiDB) might be in the intermediate states with inconsistent data. In such cases, Drainer can use the relay log to ensure that the downstream clusters are in a consistent state.

### 10.7.11.1 Consistent state during Drainer replication

The downstream clusters reaching a consistent state means the data of the downstream clusters are the same as the snapshot of the upstream which sets `tidb_snapshot = ts`.

The checkpoint consistency means Drainer checkpoint saves the consistent state of replication in `consistent`. When Drainer runs, `consistent` is `false`. After Drainer exits normally, `consistent` is set to `true`.

You can query the downstream checkpoint table as follows:

```
select * from tidb_binlog.checkpoint;
```

clusterID	checkPoint
6791641053252586769	{"consistent":false,"commitTS":414529105591271429,"ts-map":{}}

### 10.7.11.2 Implementation principles

After Drainer enables the relay log, it first writes the binlog events to the disks and then replicates the events to the downstream clusters.

If the upstream clusters are not available, Drainer can restore the downstream clusters to a consistent state by reading the relay log.

#### Note:

If the relay log data is lost at the same time, this method does not work, but its incidence is very low. In addition, you can use the Network File System to ensure data safety of the relay log.

### 10.7.11.2.1 Trigger scenarios where Drainer consumes binlogs from the relay log

When Drainer is started, if it fails to connect to the Placement Driver (PD) of the upstream clusters, and it detects that `consistent = false` in the checkpoint, Drainer will try to read the relay log, and restore the downstream clusters to a consistent state. After that, the Drainer process sets the checkpoint `consistent` to `true` and then exits.

### 10.7.11.2.2 GC mechanism of relay log

Before data is replicated to the downstream, Drainer writes data to the relay log file. If the size of a relay log file reaches 10 MB (by default) and the binlog data of the current transaction is completely written, Drainer starts to write data to the next relay log file. After Drainer successfully replicates data to the downstream, it automatically cleans up the relay log files whose data has been replicated. The relay log into which data is currently being written will not be cleaned up.

### 10.7.11.3 Configuration

To enable the relay log, add the following configuration in Drainer:

```
[syncer.relay]
### It saves the directory of the relay log. The relay log is not enabled if
    ↪ the value is empty.
### The configuration only comes to effect if the downstream is TiDB or
    ↪ MySQL.
log-dir = "/dir/to/save/log"
### The size limit of a single relay log file (unit: byte).
### When the size of a relay log file reaches this limit, data is written to
    ↪ the next relay log file.
max-file-size = 10485760
```

## 10.7.12 Bidirectional Replication between TiDB Clusters

This document describes the bidirectional replication between two TiDB clusters, how the replication works, how to enable it, and how to replicate DDL operations.

### 10.7.12.1 User scenario

If you want two TiDB clusters to exchange data changes with each other, TiDB Binlog allows you to do that. For example, you want cluster A and cluster B to replicate data with each other.

#### Note:

The data written to these two clusters must be conflict-free, that is, in the two clusters, the same primary key or the rows with the unique index of the tables must not be modified.

The user scenario is shown as below:

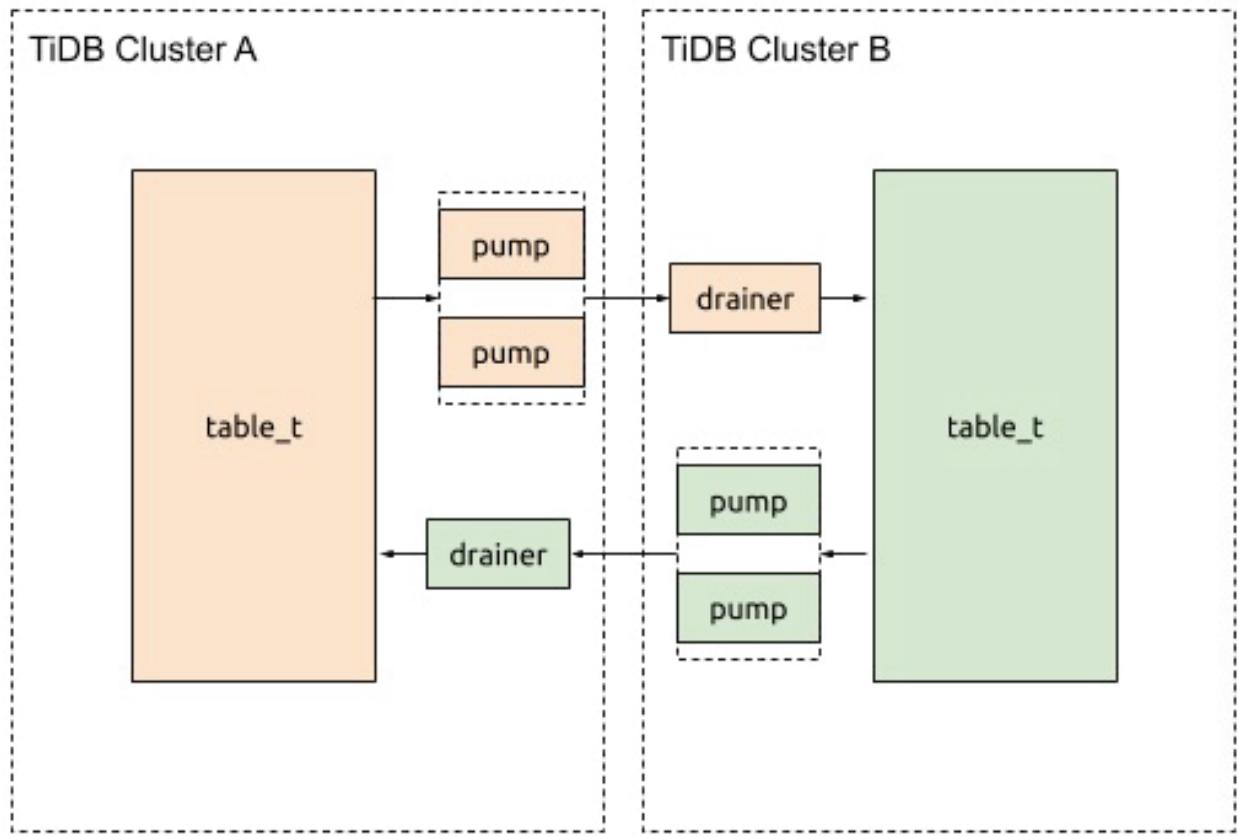


Figure 144: Architect

#### 10.7.12.2 Implementation details

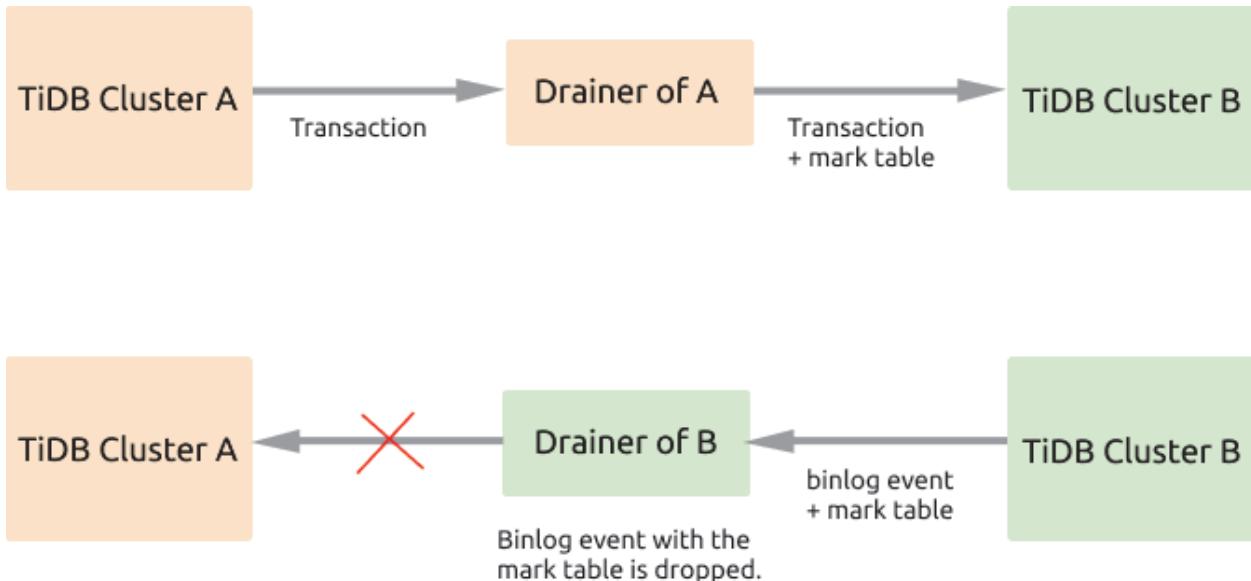


Figure 145: Mark Table

If the bidirectional replication is enabled between cluster A and cluster B, the data written to cluster A will be replicated to cluster B, and then these data changes will be replicated back to cluster A, which causes an infinite loop of replication. From the figure above, you can see that during the data replication, Drainer marks the binlog events, and filters out the marked events to avoid such a replication loop.

The detailed implementation is described as follows:

1. Start the TiDB Binlog replication program for each of the two clusters.
2. When the transaction to be replicated passes through the Drainer of cluster A, this Drainer adds the `_drainer_repl_mark` table to the transaction, writes this DML event update to the mark table, and replicate this transaction to cluster B.
3. Cluster B returns binlog events with the `_drainer_repl_mark` mark table to cluster A. The Drainer of cluster B identifies the mark table with the DML event when parsing the binlog event, and gives up replicating this binlog event to cluster A.

The replication process from cluster B to cluster A is the same as above. The two clusters can be upstream and downstream of each other.

#### Note:

- When updating the `_drainer_repl_mark` mark table, data changes are required to generate binlogs.

- DDL operations are not transactional, so you need to use the one-way replication method to replicate DDL operations. See [Replicate DDL operations](#) for details.

Drainer can use a unique ID for each connection to downstream to avoid conflicts. `channel_id` is used to indicate a channel for bidirectional replication. The two clusters should have the same `channel_id` configuration (with the same value).

If you add or delete columns in the upstream, there might be extra or missing columns of the data to be replicated to the downstream. Drainer allows this situation by ignoring the extra columns or by inserting default values to the missing columns.

#### 10.7.12.3 Mark table

The `_drainer_repl_mark` mark table has the following structure:

```
CREATE TABLE `drainer_repl_mark` (
  `id` bigint(20) NOT NULL,
  `channel_id` bigint(20) NOT NULL DEFAULT '0',
  `val` bigint(20) DEFAULT '0',
  `channel_info` varchar(64) DEFAULT NULL,
  PRIMARY KEY (`id`,`channel_id`)
);
```

Drainer uses the following SQL statement to update `_drainer_repl_mark`, which ensures data change and the generation of binlog:

```
update drainer_repl_mark set val = val + 1 where id = ? && channel_id = ?;
```

#### 10.7.12.4 Replicate DDL operations

Because Drainer cannot add the mark table to DDL operations, you can only use the one-way replication method to replicate DDL operations.

For example, if DDL replication is enabled from cluster A to cluster B, then the replication is disabled from cluster B to cluster A. This means that all DDL operations are performed on cluster A.

##### Note:

DDL operations cannot be executed on two clusters at the same time. When a DDL operation is executed, if any DML operation is being executed at the same time or any DML binlog is being replicated, the upstream and downstream table structures of the DML replication might be inconsistent.

#### 10.7.12.5 Configure and enable bidirectional replication

For bidirectional replication between cluster A and cluster B, assume that all DDL operations are executed on cluster A. On the replication path from cluster A to cluster B, add the following configuration to Drainer:

```
[syncer]
loopback-control = true
channel-id = 1 # Configures the same ID for both clusters to be replicated.
sync-ddl = true # Enables it if you need to perform DDL replication.

[syncer.to]
### 1 means SyncFullColumn and 2 means SyncPartialColumn.
### If set to SyncPartialColumn, Drainer allows the downstream table
### structure to have more or fewer columns than the data to be replicated
### And remove the STRICT_TRANS_TABLES of the SQL mode to allow fewer
    → columns, and insert zero values to the downstream.
sync-mode = 2

### Ignores the checkpoint table.
[[syncer.ignore-table]]
db-name = "tidb_binlog"
tbl-name = "checkpoint"
```

On the replication path from cluster B to cluster A, add the following configuration to Drainer:

```
[syncer]
loopback-control = true
channel-id = 1 # Configures the same ID for both clusters to be replicated.
sync-ddl = false # Disables it if you do not need to perform DDL replication
    → .

[syncer.to]
### 1 means SyncFullColumn and 2 means SyncPartialColumn.
### If set to SyncPartialColumn, Drainer allows the downstream table
### structure to have more or fewer columns than the data to be replicated
### And remove the STRICT_TRANS_TABLES of the SQL mode to allow fewer
    → columns, and insert zero values to the downstream.
sync-mode = 2

### Ignores the checkpoint table.
[[syncer.ignore-table]]
db-name = "tidb_binlog"
tbl-name = "checkpoint"
```

### 10.7.13 TiDB Binlog Glossary

This document lists the terms used in the logs, monitoring, configurations, and documentation of TiDB Binlog.

#### 10.7.13.1 Binlog

In TiDB Binlog, binlogs refer to the binary log data from TiDB. They also refer to the binary log data that Drainer writes to Kafka or files. The former and the latter are in different formats. In addition, binlogs in TiDB and binlogs in MySQL are also in different formats.

#### 10.7.13.2 Binlog event

The DML binlogs from TiDB have three types of event: `INSERT`, `UPDATE`, and `DELETE`. In the monitoring dashboard of Drainer, you can see the number of different events that correspond to the replication data.

#### 10.7.13.3 Checkpoint

A checkpoint indicates the position from which a replication task is paused and resumed, or is stopped and restarted. It records the commit-ts that Drainer replicates to the downstream. When restarted, Drainer reads the checkpoint and starts replicating data from the corresponding commit-ts.

#### 10.7.13.4 Safe mode

Safe mode refers to the mode that supports the idempotent import of DML when a primary key or unique index exists in the table schema in the incremental replication task.

In this mode, the `INSERT` statement is re-written as `REPLACE`, and the `UPDATE` statement is re-written as `DELETE` and `REPLACE`. Then the re-written statement is executed to the downstream. Safe mode is automatically enabled within 5 minutes after Drainer is started. You can manually enable the mode by modifying the `safe-mode` parameter in the configuration file, but this configuration is valid only when the downstream is MySQL or TiDB.

### 10.7.14 Troubleshoot

#### 10.7.14.1 TiDB Binlog Troubleshooting

This document describes how to troubleshoot TiDB Binlog to find the problem.

If you encounter errors while running TiDB Binlog, take the following steps to troubleshoot:

1. Check whether each monitoring metric is normal or not. Refer to [TiDB Binlog Monitoring](#) for details.

2. Use the [binlogctl tool](#) to check whether the state of each Pump or Drainer node is normal or not.
3. Check whether `ERROR` or `WARN` exists in the Pump log or Drainer log.

After finding out the problem by the above steps, refer to [FAQ](#) and [TiDB Binlog Error Handling](#) for the solution. If you fail to find the solution or the solution provided does not help, submit an [issue](#) for help.

#### 10.7.14.2 TiDB Binlog Error Handling

This document introduces common errors that you might encounter and solutions to these errors when you use TiDB Binlog.

##### 10.7.14.2.1 `kafka server: Message was too large, server rejected it to avoid allocation error is returned when Drainer replicates data to Kafka`

Cause: Executing a large transaction in TiDB generates binlog data of a large size, which might exceed Kafka's limit on the message size.

Solution: Adjust the configuration parameters of Kafka as shown below:

```
message.max.bytes=1073741824  
replica.fetch.max.bytes=1073741824  
fetch.message.max.bytes=1073741824
```

##### 10.7.14.2.2 `Pump returns no space left on device error`

Cause: The local disk space is insufficient for Pump to write binlog data normally.

Solution: Clean up the disk space and then restart Pump.

##### 10.7.14.2.3 `fail to notify all living drainer is returned when Pump is started`

Cause: When Pump is started, it notifies all Drainer nodes that are in the `online` state. If it fails to notify Drainer, this error log is printed.

Solution: Use the [binlogctl tool](#) to check whether each Drainer node is normal or not. This is to ensure that all Drainer nodes that are in the `online` state are working normally. If the state of a Drainer node is not consistent with its actual working status, use the `binlogctl` tool to change its state and then restart Pump.

##### 10.7.14.2.4 `Data loss occurs during the TiDB Binlog replication`

You need to confirm that TiDB Binlog is enabled on all TiDB instances and runs normally. If the cluster version is later than v3.0, use the `curl {TiDB_IP}:{STATUS_PORT}/→ info/all` command to confirm the TiDB Binlog status on all TiDB instances.

#### 10.7.14.2.5 When the upstream transaction is large, Pump reports an error

```
rpc error: code = ResourceExhausted desc = trying to send message larger than max (2191430008 vs. 2147483647)
```

This error occurs because the gRPC message sent by TiDB to Pump exceeds the size limit. You can adjust the maximum size of a gRPC message that Pump allows by specifying `max-message-size` when starting Pump.

#### 10.7.14.2.6 Is there any cleaning mechanism for the incremental data of the file format output by Drainer? Will the data be deleted?

- In Drainer v3.0.x, there is no cleaning mechanism for incremental data of the file format.
- In the v4.0.x version, there is a time-based data cleaning mechanism. For details, refer to [Drainer's retention-time configuration item](#).

### 10.7.15 TiDB Binlog FAQ

This document collects the frequently asked questions (FAQs) about TiDB Binlog.

#### 10.7.15.1 What is the impact of enabling TiDB Binlog on the performance of TiDB?

- There is no impact on the query.
- There is a slight performance impact on `INSERT`, `DELETE` and `UPDATE` transactions. In latency, a p-binlog is written concurrently in the TiKV prewrite stage before the transactions are committed. Generally, writing binlog is faster than TiKV prewrite, so it does not increase latency. You can check the response time of writing binlog in Pump's monitoring panel.

#### 10.7.15.2 How high is the replication latency of TiDB Binlog?

The latency of TiDB Binlog replication is measured in seconds, which is generally about 3 seconds during off-peak hours.

#### 10.7.15.3 What privileges does Drainer need to replicate data to the downstream MySQL or TiDB cluster?

To replicate data to the downstream MySQL or TiDB cluster, Drainer must have the following privileges:

- Insert
- Update

- Delete
- Create
- Drop
- Alter
- Execute
- Index
- Select

#### 10.7.15.4 What can I do if the Pump disk is almost full?

1. Check whether Pump's GC works well:
  - Check whether the **gc\_tso** time in Pump's monitoring panel is identical with that of the configuration file.
2. If GC works well, perform the following steps to reduce the amount of space required for a single Pump:
  - Modify the **GC** parameter of Pump to reduce the number of days to retain data.
  - Add pump instances.

#### 10.7.15.5 What can I do if Drainer replication is interrupted?

Execute the following command to check whether the status of Pump is normal and whether all the Pump instances that are not in the **offline** state are running.

```
binlogctl -cmd pumps
```

Then, check whether the Drainer monitor or log outputs corresponding errors. If so, resolve them accordingly.

#### 10.7.15.6 What can I do if Drainer is slow to replicate data to the downstream MySQL or TiDB cluster?

Check the following monitoring items:

- For the **Drainer Event** monitoring metric, check the speed of Drainer replicating **INSERT**, **UPDATE** and **DELETE** transactions to the downstream per second.
- For the **SQL Query Time** monitoring metric, check the time Drainer takes to execute SQL statements in the downstream.

Possible causes and solutions for slow replication:

- If the replicated database contains a table without a primary key or unique index, add a primary key to the table.

- If the latency between Drainer and the downstream is high, increase the value of the `worker-count` parameter of Drainer. For cross-datacenter replication, it is recommended to deploy Drainer in the downstream.
- If the load in the downstream is not high, increase the value of the `worker-count` parameter of Drainer.

#### 10.7.15.7 What can I do if a Pump instance crashes?

If a Pump instance crashes, Drainer cannot replicate data to the downstream because it cannot obtain the data of this instance. If this Pump instance can recover to the normal state, Drainer resumes replication; if not, perform the following steps:

1. Use `binlogctl` to change the state of this Pump instance to `offline` to discard the data of this Pump instance.
2. Because Drainer cannot obtain the data of this pump instance, the data in the downstream and upstream is inconsistent. In this situation, perform full and incremental backups again. The steps are as follows:
  1. Stop the Drainer.
  2. Perform a full backup in the upstream.
  3. Clear the data in the downstream including the `tidb_binlog.checkpoint` table.
  4. Restore the full backup to the downstream.
  5. Deploy Drainer and use `initialCommitTs` (set `initialCommitTs` as the snapshot timestamp of the full backup) as the start point of initial replication.

#### 10.7.15.8 What is checkpoint?

Checkpoint records the `commit-ts` that Drainer replicates to the downstream. When Drainer restarts, it reads the checkpoint and then replicates data to the downstream starting from the corresponding `commit-ts`. The `["write save point"] [ts ↦ =411222863322546177]` Drainer log means saving the checkpoint with the corresponding timestamp.

Checkpoint is saved in different ways for different types of downstream platforms:

- For MySQL/TiDB, it is saved in the `tidb_binlog.checkpoint` table.
- For Kafka/file, it is saved in the file of the corresponding configuration directory.

The data of kafka/file contains `commit-ts`, so if the checkpoint is lost, you can check the latest `commit-ts` of the downstream data by consuming the latest data in the downstream .

Drainer reads the checkpoint when it starts. If Drainer cannot read the checkpoint, it uses the configured `initialCommitTs` as the start point of the initial replication.

#### 10.7.15.9 How to redeploy Drainer on the new machine when Drainer fails and the data in the downstream remains?

If the data in the downstream is not affected, you can redeploy Drainer on the new machine as long as the data can be replicated from the corresponding checkpoint.

- If the checkpoint is not lost, perform the following steps:
  1. Deploy and start a new Drainer (Drainer can read checkpoint and resumes replication).
  2. Use `binlogctl` to change the state of the old Drainer to `offline`.
- If the checkpoint is lost, perform the following steps:
  1. To deploy a new Drainer, obtain the `commit-ts` of the old Drainer as the `initialCommitTs` of the new Drainer.
  2. Use `binlogctl` to change the state of the old Drainer to `offline`.

#### 10.7.15.10 How to restore the data of a cluster using a full backup and a binlog backup file?

1. Clean up the cluster and restore a full backup.
2. To restore the latest data of the backup file, use Reparo to set `start-tso = {snapshot timestamp of the full backup + 1}` and `end-ts = 0` (or you can specify a point in time).

#### 10.7.15.11 How to redeploy Drainer when enabling `ignore-error` in Primary-Secondary replication triggers a critical error?

If a critical error is triggered when TiDB fails to write binlog after enabling `ignore-error`, TiDB stops writing binlog and binlog data loss occurs. To resume replication, perform the following steps:

1. Stop the Drainer instance.
2. Restart the `tidb-server` instance that triggers critical error and resume writing binlog (TiDB does not write binlog to Pump after critical error is triggered).
3. Perform a full backup in the upstream.
4. Clear the data in the downstream including the `tidb_binlog.checkpoint` table.
5. Restore the full backup to the downstream.
6. Deploy Drainer and use `initialCommitTs` (set `initialCommitTs` as the snapshot timestamp of the full backup) as the start point of initial replication.

#### 10.7.15.12 When can I pause or close a Pump or Drainer node?

Refer to [TiDB Binlog Cluster Operations](#) to learn the description of the Pump or Drainer state and how to start and exit the process.

Pause a Pump or Drainer node when you need to temporarily stop the service. For example:

- Version upgrade

Use the new binary to restart the service after the process is stopped.

- Server maintenance

When the server needs a downtime maintenance, exit the process and restart the service after the maintenance is finished.

Close a Pump or Drainer node when you no longer need the service. For example:

- Pump scale-in

If you do not need too many Pump services, close some of them.

- Cancelling replication tasks

If you no longer need to replicate data to a downstream database, close the corresponding Drainer node.

- Service migration

If you need to migrate the service to another server, close the service and re-deploy it on the new server.

#### 10.7.15.13 How can I pause a Pump or Drainer process?

- Directly kill the process.

**Note:**

Do not use the `kill -9` command. Otherwise, the Pump or Drainer node cannot process signals.

- If the Pump or Drainer node runs in the foreground, pause it by pressing Ctrl+C.
- Use the `pause-pump` or `pause-drainer` command in binlogctl.

#### 10.7.15.14 Can I use the `update-pump` or `update-drainer` command in binlogctl to pause the Pump or Drainer service?

No. The `update-pump` or `update-drainer` command directly modifies the state information saved in PD without notifying Pump or Drainer to perform the corresponding operation. Misusing the two commands can interrupt data replication and might even cause data loss.

#### 10.7.15.15 Can I use the `update-pump` or `update-drainer` command in binlogctl to close the Pump or Drainer service?

No. The `update-pump` or `update-drainer` command directly modifies the state information saved in PD without notifying Pump or Drainer to perform the corresponding operation. Misusing the two commands interrupts data replication and might even cause data inconsistency. For example:

- When a Pump node runs normally or is in the `paused` state, if you use the `update-pump` command to set the Pump state to `offline`, the Drainer node stops pulling the binlog data from the `offline` Pump. In this situation, the newest binlog cannot be replicated to the Drainer node, causing data inconsistency between upstream and downstream.
- When a Drainer node runs normally, if you use the `update-drainer` command to set the Drainer state to `offline`, the newly started Pump node only notifies Drainer nodes in the `online` state. In this situation, the `offline` Drainer fails to pull the binlog data from the Pump node in time, causing data inconsistency between upstream and downstream.

#### 10.7.15.16 When can I use the `update-pump` command in binlogctl to set the Pump state to `paused`?

In some abnormal situations, Pump fails to correctly maintain its state. Then, use the `update-pump` command to modify the state.

For example, when a Pump process is exited abnormally (caused by directly exiting the process when a panic occurs or mistakenly using the `kill -9` command to kill the process), the Pump state information saved in PD is still `online`. In this situation, if you do not need to restart Pump to recover the service at the moment, use the `update-pump` command to update the Pump state to `paused`. Then, interruptions can be avoided when TiDB writes binlogs and Drainer pulls binlogs.

#### 10.7.15.17 When can I use the `update-drainer` command in binlogctl to set the Drainer state to `paused`?

In some abnormal situations, the Drainer node fails to correctly maintain its state, which has influenced the replication task. Then, use the `update-drainer` command to modify the state.

For example, when a Drainer process is exited abnormally (caused by directly exiting the process when a panic occurs or mistakenly using the `kill -9` command to kill the process), the Drainer state information saved in PD is still `online`. When a Pump node is started, it fails to notify the exited Drainer node (the `notify drainer ... error`), which cause the Pump node failure. In this situation, use the `update-drainer` command to update the Drainer state to `paused` and restart the Pump node.

#### 10.7.15.18 How can I close a Pump or Drainer node?

Currently, you can only use the `offline-pump` or `offline-drainer` command in binlogctl to close a Pump or Drainer node.

#### 10.7.15.19 When can I use the `update-pump` command in binlogctl to set the Pump state to `offline`?

You can use the `update-pump` command to set the Pump state to `offline` in the following situations:

- When a Pump process is exited abnormally and the service cannot be recovered, the replication task is interrupted. If you want to recover the replication and accept some losses of binlog data, use the `update-pump` command to set the Pump state to `offline` → . Then, the Drainer node stops pulling binlog from the Pump node and continues replicating data.
- Some stale Pump nodes are left over from historical tasks. Their processes have been exited and their services are no longer needed. Then, use the `update-pump` command to set their state to `offline`.

For other situations, use the `offline-pump` command to close the Pump service, which is the regular process.

##### Warning:

Do not use the `update-pump` command unless you can tolerate binlog data loss and data inconsistency between upstream and downstream, or you no longer need the binlog data stored in the Pump node.

#### 10.7.15.20 Can I use the `update-pump` command in binlogctl to set the Pump state to `offline` if I want to close a Pump node that is exited and set to `paused`?

When a Pump process is exited and the node is in the paused state, not all the binlog data in the node is consumed in its downstream Drainer node. Therefore, doing so might risk data inconsistency between upstream and downstream. In this situation, restart the Pump and use the `offline-pump` command to close the Pump node.

#### 10.7.15.21 When can I use the `update-drainer` command in `binlogctl` to set the Drainer state to `offline`?

Some stale Drainer nodes are left over from historical tasks. Their processes have been exited and their services are no longer needed. Then, use the `update-drainer` command to set their state to `offline`.

#### 10.7.15.22 Can I use SQL operations such as `change pump` and `change drainer` to pause or close the Pump or Drainer service?

No. For more details on these SQL operations, refer to [Use SQL statements to manage Pump or Drainer](#).

These SQL operations directly modifies the state information saved in PD and are functionally equivalent to the `update-pump` and `update-drainer` commands in `binlogctl`. To pause or close the Pump or Drainer service, use the `binlogctl` tool.

#### 10.7.15.23 What can I do when some DDL statements supported by the upstream database cause error when executed in the downstream database?

To solve the problem, follow these steps:

1. Check `drainer.log`. Search `exec failed` for the last failed DDL operation before the Drainer process is exited.
2. Change the DDL version to the one compatible to the downstream. Perform this step manually in the downstream database.
3. Check `drainer.log`. Search for the failed DDL operation and find the `commit-ts` of this operation. For example:

```
[2020/05/21 09:51:58.019 +08:00] [INFO] [syncer.go:398] ["add ddl item
    ↳ to syncer, you can add this commit ts to `ignore-txn-commit-ts`"
    ↳ to skip this ddl if needed"] [sql="ALTER TABLE `test` ADD INDEX
    ↳ (`index1`)"] [{"commit_ts":416815754209656834}].
```

4. Modify the `drainer.toml` configuration file. Add the `commit-ts` in the `ignore-txn-commit-ts` item and restart the Drainer node.

#### 10.7.15.24 TiDB fails to write to binlog and gets stuck, and `listener stopped, waiting for manual stop` appears in the log

In TiDB v3.0.12 and earlier versions, the binlog write failure causes TiDB to report the fatal error. TiDB does not automatically exit but only stops the service, which seems like getting stuck. You can see the `listener stopped, waiting for manual stop` error in the log.

You need to determine the specific causes of the binlog write failure. If the failure occurs because binlog is slowly written into the downstream, you can consider scaling out Pump or increasing the timeout time for writing binlog.

Since v3.0.13, the error-reporting logic is optimized. The binlog write failure causes transaction execution to fail and TiDB Binlog will return an error but will not get TiDB stuck.

#### 10.7.15.25 TiDB writes duplicate binlogs to Pump

This issue does not affect the downstream and replication logic.

When the binlog write fails or becomes timeout, TiDB retries writing binlog to the next available Pump node until the write succeeds. Therefore, if the binlog write to a Pump node is slow and causes TiDB timeout (default 15s), then TiDB determines that the write fails and tries to write to the next Pump node. If binlog is actually successfully written to the timeout-causing Pump node, the same binlog is written to multiple Pump nodes. When Drainer processes the binlog, it automatically de-duplicates binlogs with the same TSO, so this duplicate write does not affect the downstream and replication logic.

#### 10.7.15.26 Reparo is interrupted during the full and incremental restore process. Can I use the last TSO in the log to resume replication?

Yes. Reparo does not automatically enable the safe-mode when you start it. You need to perform the following steps manually:

1. After Reparo is interrupted, record the last TSO in the log as `checkpoint-tso`.
2. Modify the Reparo configuration file, set the configuration item `start-tso` to `checkpoint-tso + 1`, set `stop-tso` to `checkpoint-tso + 80,000,000,000` (approximately five minutes after the `checkpoint-tso`), and set `safe-mode` to `true`. Start Reparo, and Reparo replicates data to `stop-tso` and then stops automatically.
3. After Reparo stops automatically, set `start-tso` to `checkpoint tso + 80,000,000,001 ↵`, set `stop-tso` to 0, and set `safe-mode` to `false`. Start Reparo to resume replication.

## 10.8 TiDB Lightning

### 10.8.1 TiDB Lightning Overview

TiDB Lightning is a tool used for fast full import of large amounts of data into a TiDB

cluster. You can download TiDB Lightning from [here](#).

Currently, TiDB Lightning can mainly be used in the following two scenarios:

- Importing **large amounts** of **new** data **quickly**
- Restore all backup data

Currently, TiDB Lightning supports:

- The data source of the [Dumpling](#), CSV or [Amazon Aurora Parquet](#) exported formats.
- Reading data from a local disk or from the Amazon S3 storage. For details, see [External Storages](#).

#### 10.8.1.1 TiDB Lightning architecture

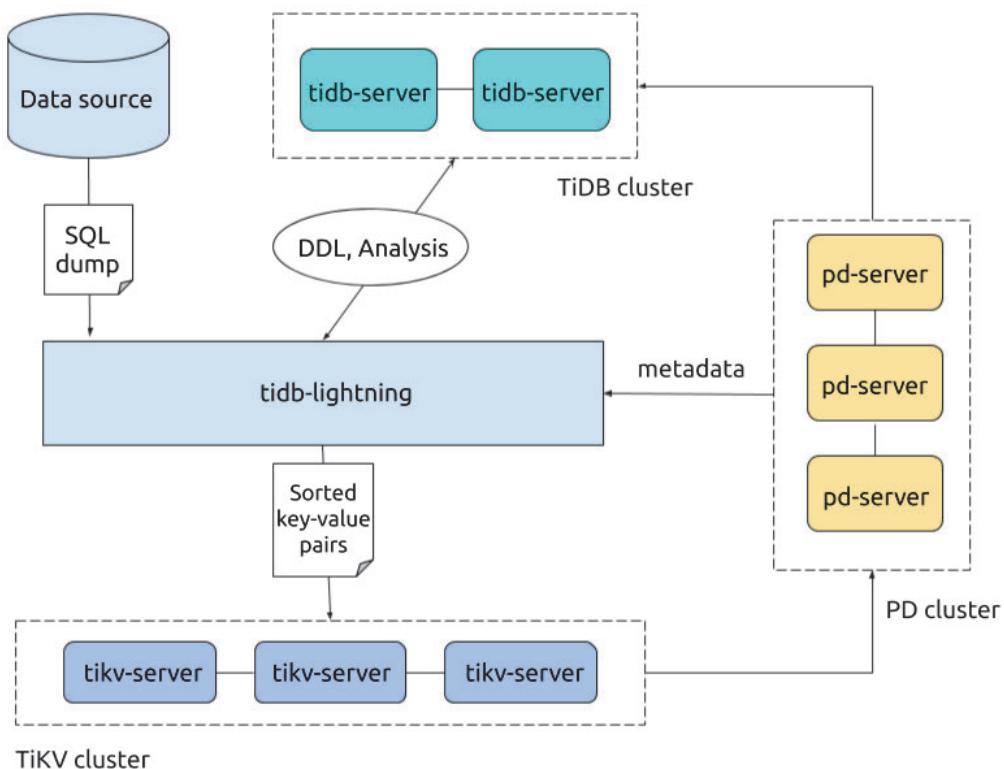


Figure 146: Architecture of TiDB Lightning tool set

The complete import process is as follows:

1. Before importing, `tidb-lightning` switches the TiKV cluster to “import mode”, which optimizes the cluster for writing and disables automatic compaction.
2. `tidb-lightning` creates the skeleton of all tables from the data source.
3. Each table is split into multiple continuous *batches*, so that data from a huge table (200 GB+) can be imported incrementally and concurrently.
4. For each batch, `tidb-lightning` creates an *engine file* to store KV pairs. `tidb-lightning` then reads the data source in parallel, transforms each row into KV pairs according to the TiDB rules, and writes these KV pairs into the local files for temporary storage.
5. Once a complete engine file is written, `tidb-lightning` divides and schedules these data and imports them into the target TiKV cluster.

There are two kinds of engine files: *data engines* and *index engines*, each corresponding to two kinds of KV pairs: the row data and secondary indices. Normally, the row data are entirely sorted in the data source, while the secondary indices are out of order. Because of this, the data engines are uploaded as soon as a batch is completed, while the index engines are imported only after all batches of the entire table are encoded.

6. After all engines associated to a table are imported, `tidb-lightning` performs a checksum comparison between the local data source and those calculated from the cluster, to ensure there is no data corruption in the process; tells TiDB to `ANALYZE` all imported tables, to prepare for optimal query planning; and adjusts the `AUTO_INCREMENT` value so future insertions will not cause conflict.

The auto-increment ID of a table is computed by the estimated *upper bound* of the number of rows, which is proportional to the total file size of the data files of the table. Therefore, the final auto-increment ID is often much larger than the actual number of rows. This is expected since in TiDB auto-increment is **not necessarily allocated sequentially**.

7. Finally, `tidb-lightning` switches the TiKV cluster back to “normal mode”, so the cluster resumes normal services.

If the target cluster of data import is v3.x or earlier versions, you need to use the Importer-backend to import data. In this mode, `tidb-lightning` sends the parsed KV pairs to `tikv-importer` via gRPC and `tikv-importer` imports the data.

TiDB Lightning also supports using TiDB-backend for data import. In this mode, `tidb-lightning` transforms data into `INSERT` SQL statements and directly executes them on the target cluster. See [TiDB Lightning Backends](#) for details.

### 10.8.1.2 Restrictions

If you use TiDB Lighting together with TiFlash:

No matter a table has TiFlash replica(s) or not, you can import data to that table using TiDB Lightning. Note that this might slow the TiDB Lightning procedure, which depends on the NIC bandwidth on the lightning host, the CPU and disk load of the TiFlash node, and the number of TiFlash replicas.

### 10.8.2 TiDB Lightning Tutorial

[TiDB Lightning](#) is a tool used for fast full import of large amounts of data into a TiDB cluster. Currently, TiDB Lightning supports reading SQL dump exported via SQL or CSV data source. You can use it in the following two scenarios:

- Import **large amounts of new data quickly**
- Back up and restore all the data

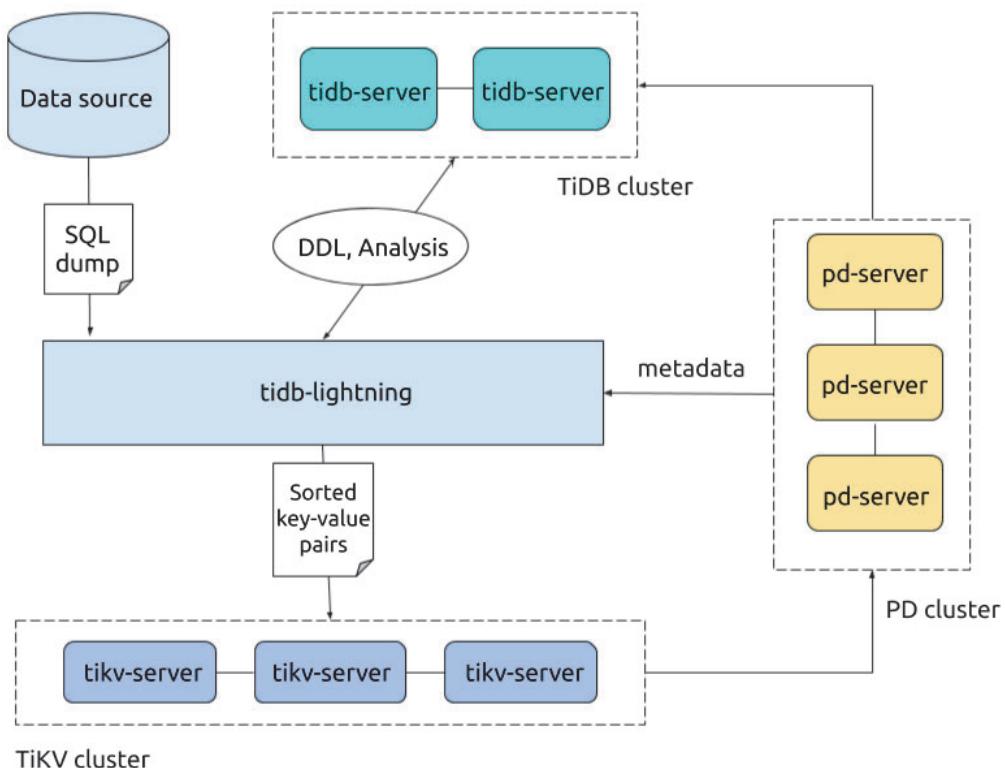


Figure 147: Architecture of TiDB Lightning tool set

### 10.8.2.1 Prerequisites

This tutorial assumes you use several new and clean CentOS 7 instances. You can use VMware, VirtualBox or other tools to deploy a virtual machine locally or a small cloud virtual machine on a vendor-supplied platform. Because TiDB Lightning consumes a large amount of computer resources, it is recommended that you allocate at least 16 GB memory and CPU of 32 cores for running it with the best performance.

#### Warning:

The deployment method in this tutorial is only recommended for test and trial. **Do not apply it in the production or development environment.**

### 10.8.2.2 Prepare full backup data

First, use `dumpling` to export data from MySQL:

```
./bin/dumpling -h 127.0.0.1 -P 3306 -u root -t 16 -F 256MB -B test -f 'test  
↪ .t[12]' -o /data/my_database/
```

In the above command:

- `-B test`: means the data is exported from the `test` database.
- `-f test.t[12]`: means only the `test.t1` and `test.t2` tables are exported.
- `-t 16`: means 16 threads are used to export the data.
- `-F 256MB`: means a table is partitioned into chunks and one chunk is 256 MB.

After executing this command, the full backup data is exported to the `/data/`  
`↪ my_database` directory.

### 10.8.2.3 Deploy TiDB Lightning

#### 10.8.2.3.1 Step 1: Deploy TiDB cluster

Before the data import, you need to deploy a TiDB cluster. In this tutorial, TiDB v5.0.0 is used as an example. For the deployment method, refer to [Deploy a TiDB Cluster Using TiUP](#).

#### 10.8.2.3.2 Step 2: Download TiDB Lightning installation package

Download the TiDB Lightning installation package from the following link:

- v5.0.0: [tidb-toolkit-v5.0.0-linux-amd64.tar.gz](#)

**Note:**

TiDB Lightning is compatible with TiDB clusters of earlier versions. It is recommended that you download the latest stable version of the TiDB Lightning installation package.

#### 10.8.2.3.3 Step 3: Start `tidb-lightning`

1. Upload `bin/tidb-lightning` and `bin/tidb-lightning-ctl` in the package to the server where TiDB Lightning is deployed.
2. Upload the [prepared data source](#) to the server.
3. Configure `tidb-lightning.toml` as follows:

```
[lightning]
# Logging
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
# Uses the Local-backend
backend = "local"
# Sets the directory for temporarily storing the sorted key-value pairs
    ↪ .
# The target directory must be empty.
sorted-kv-dir = "/mnt/ssd/sorted-kv-dir"

[mydumper]
# Local source data directory
data-source-dir = "/data/my_datasource/"

# Configures the wildcard rule. By default, all tables in the mysql,
    ↪ sys, INFORMATION_SCHEMA, PERFORMANCE_SCHEMA, METRICS_SCHEMA, and
    ↪ INSPECTION_SCHEMA system databases are filtered.
# If this item is not configured, the "cannot find schema" error occurs
    ↪ when system tables are imported.
filter = ['*.*', '!mysql.*', '!sys.*', '!INFORMATION_SCHEMA.*', '!
    ↪ PERFORMANCE_SCHEMA.*', '!METRICS_SCHEMA.*', '!INSPECTION_SCHEMA
    ↪ .*']
```

```
[tidb]
# Information of the target cluster
host = "172.16.31.2"
port = 4000
user = "root"
password = "rootroot"
# Table schema information is fetched from TiDB via this status-port.
status-port = 10080
# The PD address of the cluster
pd-addr = "172.16.31.3:2379"
```

- After configuring the parameters properly, use a `nohup` command to start the `tidb → lightning` process. If you directly run the command in the command-line, the process might exit because of the SIGHUP signal received. Instead, it's preferable to run a bash script that contains the `nohup` command:

```
#!/bin/bash
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

#### 10.8.2.3.4 Step 4: Check data integrity

After the import is completed, TiDB Lightning exits automatically. If the import is successful, you can find `tidb lightning exit` in the last line of the log file.

If any error occurs, refer to [TiDB Lightning FAQs](#).

#### 10.8.2.4 Summary

This tutorial briefly introduces what TiDB Lightning is and how to quickly deploy a TiDB Lightning cluster to import full backup data to the TiDB cluster.

For detailed features and usage about TiDB Lightning, refer to [TiDB Lightning Overview](#).

### 10.8.3 TiDB Lightning Deployment

This document describes the hardware requirements of TiDB Lightning using the Local-backend, and how to deploy it manually.

If Local-backend is used for data import, during the import process, **the cluster cannot provide services**. If you do not want the TiDB services to be impacted, perform the data import according to [TiDB Lightning TiDB-backend](#).

#### 10.8.3.1 Notes

Before starting TiDB Lightning, note that:

- If `tidb-lightning` crashes, the cluster is left in “import mode”. Forgetting to switch back to “normal mode” can lead to a high amount of uncompacted data on the TiKV cluster, and cause abnormally high CPU usage and stall. You can manually switch the cluster back to “normal mode” via the `tidb-lightning-ctl` tool:

```
bin/tidb-lightning-ctl --switch-mode=normal
```

- TiDB Lightning is required to have the following privileges in the downstream TiDB:

Privilege	Scope
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables

If the `checksum` configuration item of TiDB Lightning is set to `true`, then the admin user privileges in the downstream TiDB need to be granted to TiDB Lightning.

#### 10.8.3.2 Hardware requirements

`tidb-lightning` is a resource-intensive program. It is recommended to deploy it as follows.

- 32+ logical cores CPU
- 20GB+ memory
- An SSD large enough to store the entire data source, preferring higher read speed
- 10 Gigabit network card (capable of transferring at 1 GB/s)
- `tidb-lightning` fully consumes all CPU cores when running, and deploying on a dedicated machine is highly recommended. If not possible, `tidb-lightning` could be deployed together with other components like `tidb-server`, and the CPU usage could be limited via the `region-concurrency` setting.

#### Note:

- `tidb-lightning` is a CPU intensive program. In an environment with mixed components, the resources allocated to `tidb-lightning` must be limited. Otherwise, other components might not be able to run. It is recommended to set the `region-concurrency` to 75% of CPU logical cores. For instance, if the CPU has 32 logical cores, you can set the `region-concurrency` to 24.

Additionally, the target TiKV cluster should have enough space to absorb the new data. Besides [the standard requirements](#), the total free space of the target TiKV cluster should be larger than **Size of data source × Number of replicas × 2**.

With the default replica count of 3, this means the total free space should be at least 6 times the size of data source.

#### 10.8.3.3 Export data

Use the [dumpling tool](#) to export data from MySQL by using the following command:

```
./bin/dumpling -h 127.0.0.1 -P 3306 -u root -t 16 -F 256MB -B test -f 'test  
↪ .t[12]' -o /data/my_database/
```

In this command,

- **-B test**: means the data is exported from the `test` database.
- **-f test.t[12]**: means only the `test.t1` and `test.t2` tables are exported.
- **-t 16**: means 16 threads are used to export the data.
- **-F 256MB**: means a table is partitioned into chunks and one chunk is 256 MB.

If the data source consists of CSV files, see [CSV support](#) for configuration.

#### 10.8.3.4 Deploy TiDB Lightning

This section describes how to [deploy TiDB Lightning manually](#).

##### 10.8.3.4.1 Deploy TiDB Lightning manually

Step 1: Deploy a TiDB cluster

Before importing data, you need to have a deployed TiDB cluster. It is highly recommended to use the latest stable version.

You can find deployment instructions in [TiDB Quick Start Guide](#).

Step 2: Download the TiDB Lightning installation package

Refer to the [TiDB enterprise tools download page](#) to download the TiDB Lightning package.

##### Note:

TiDB Lightning is compatible with TiDB clusters of earlier versions. It is recommended that you download the latest stable version of the TiDB Lightning installation package.

### Step 3: Start tidb-lightning

1. Upload bin/tidb-lightning and bin/tidb-lightning-ctl from the tool set.
2. Mount the data source onto the same machine.
3. Configure `tidb-lightning.toml`. For configurations that do not appear in the template below, TiDB Lightning writes a configuration error to the log file and exits. `sorted-kv-dir` must be an empty directory and the disk where the directory is located must have a lot of free space.

```
[lightning]
# The concurrency number of data. It is set to the number of logical
# cores by default. When deploying together with other components, you
# set it to 75% of the size of logical CPU cores to limit the CPU usage
# .
# region-concurrency =

# Logging
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
# Sets the backend to the "local" mode.
backend = "local"
# Sets the directory of temporary local storage.
sorted-kv-dir = "/mnt/ssd/sorted-kv-dir"

[mydumper]
# Local source data directory
data-source-dir = "/data/my_database"

[tidb]
# Configuration of any TiDB server from the cluster
host = "172.16.31.1"
port = 4000
user = "root"
password = ""
# Table schema information is fetched from TiDB via this status-port.
status-port = 10080
# An address of pd-server.
pd-addr = "172.16.31.4:2379"
```

The above only shows the essential settings. See the [Configuration](#) section for the full list of settings.

4. Run `tidb-lightning`.

```
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

#### 10.8.3.5 Upgrading TiDB Lightning

You can upgrade TiDB Lightning by replacing the binaries alone. No further configuration is needed. See [FAQ](#) for the detailed instructions of restarting TiDB Lightning.

If an import task is running, we recommend you to wait until it finishes before upgrading TiDB Lightning. Otherwise, there might be chances that you need to reimport from scratch, because there is no guarantee that checkpoints work across versions.

#### 10.8.4 TiDB Lightning Prechecks

Starting from TiDB 5.3.0, TiDB Lightning provides the ability to check the configuration before running a migration task. It is enabled by default. This feature automatically performs some routine checks for disk space and execution configuration. The main purpose is to ensure that the whole subsequent import process goes smoothly.

The following table describes each check item and detailed explanation.

Check	Items	Description
Cluster Check	ver- sion and sta- tus	whether the clus- ter can be con- nected in the con- figu- ra- tion, and whether the TiKV/PD/Ti- Flash ver- sion sup- ports the Local im- port mode when the back- end mode is Lo- cal.

Check	Items	Description
Disk	Check space	whether there is enough space on the local disk and on the TiKV cluster for import-ing data.
TiDB	Light-ning samples	sources and esti-mates the per-cent-age of the index size from the sample re-sult.

---

Check	
Items	Description
Region	Check
dis-	whether
tribu-	the
tion	re-
sta-	gions
tus	in
	the
	TiKV
	clus-
	ter
	are
	dis-
	tributed
	evenly
	and
	whether
	there
	are
	too
	many
	empty
	re-
	gions.
	If the
	num-
	ber
	of
	empty
	re-
	gions
	ex-
	ceeds
	$\max(1000,$
	num-
	ber
	of ta-
	bles
	$* 3),$
	i.e. greater
	than
	the
	big-
	ger
	one
1073	of
	“1000”
	or “3
	“;

---

---

Check  
Items   Description

ExceedingWhen  
Large there  
CSV are  
files CSV  
in files  
the larger  
data than  
file 10  
GiB  
in  
the  
backup  
file  
and  
auto-  
slicing  
is  
not  
en-  
abled  
(Strict-  
For-  
mat=false),  
it  
will  
im-  
pact  
the  
im-  
port  
per-  
for-  
mance.  
The  
pur-  
pose  
of  
this  
check  
is to  
re-  
mind  
you  
to en-  
sure  
the  
data  
is i

---

Check	Items	Description
RecoverThis from check break- en- points sures that no changes are made to the source file or schema in the database dur- ing the break- point re- cov- ery pro- cess that would re- sult in im- port- ing the wrong data.		

---

Check  
Items   Description

Import When  
into im-  
an port-  
exist- ing  
ing into  
table an al-  
ready  
cre-  
ated  
ta-  
ble,  
it  
checks,  
as  
much  
as  
possi-  
ble,  
whether  
the  
source  
file  
matches  
the  
exist-  
ing  
ta-  
ble.  
Check  
if the  
num-  
ber  
of  
columns  
matches.  
If the  
source  
file  
has  
col-  
umn  
names,  
check  
if the  
1076 col-  
umn  
names

Check	Items	Description
-------	-------	-------------

## 10.8.5 TiDB Lightning Configuration

This document provides samples for global configuration, task configuration, and TiKV Importer configuration in TiDB Lightning, and describes the usage of command-line parameters.

### 10.8.5.1 Configuration files

TiDB Lightning has two configuration classes: “global” and “task”, and they have compatible structures. Their distinction arises only when the `server mode` is enabled. When server mode is disabled (the default), TiDB Lightning will only execute one task, and the same configuration file is used for both global and task configurations.

#### 10.8.5.1.1 TiDB Lightning (Global)

```
##### tidb-lightning global configuration

[lightning]
### The HTTP port for the web interface and Prometheus metrics pulling (0 to
→ disable).
status-addr = ':8289'

### Toggle server mode and use of the web interface.
### See the "TiDB Lightning Web Interface" section for details.
server-mode = false

### Logging
level = "info"
file = "tidb-lightning.log"
max-size = 128 # MB
max-days = 28
max-backups = 14
```

#### 10.8.5.1.2 TiDB Lightning (Task)

```
##### tidb-lightning task configuration

[lightning]
### Checks whether the cluster satisfies the minimum requirement before
→ starting.
#check-requirements = true
```

```

### The maximum number of engines to be opened concurrently.
### Each table is split into one "index engine" to store indices, and
    ↪ multiple
### "data engines" to store row data. These settings control the maximum
### concurrent number for each type of engines.
### These values affect the memory and disk usage of tikv-importer.
### The sum of these two values must not exceed the max-open-engines setting
### for tikv-importer.
index-concurrency = 2
table-concurrency = 6

### The concurrency number of data. It is set to the number of logical CPU
### cores by default. When deploying together with other components, you can
### set it to 75% of the size of logical CPU cores to limit the CPU usage.
#region-concurrency =

### The maximum I/O concurrency. Excessive I/O concurrency causes an
    ↪ increase in
### I/O latency because the disk's internal buffer is frequently refreshed,
### which causes the cache miss and slows down the read speed. Depending on
    ↪ the storage
### medium, this value might need to be adjusted for optimal performance.
io-concurrency = 5

[security]
### Specifies certificates and keys for TLS connections within the cluster.
### Public certificate of the CA. Leave empty to disable TLS.
### ca-path = "/path/to/ca.pem"
### Public certificate of this service.
### cert-path = "/path/to/lightning.pem"
### Private key of this service.
### key-path = "/path/to/lightning.key"

[checkpoint]
### Whether to enable checkpoints.
### While importing data, TiDB Lightning records which tables have been
    ↪ imported, so
### even if TiDB Lightning or another component crashes, you can start from
    ↪ a known
### good state instead of restarting from scratch.
enable = true
### The schema name (database name) to store the checkpoints.
schema = "tidb_lightning_checkpoint"
### Where to store the checkpoints.

```

```

### - file: store as a local file.
### - mysql: store into a remote MySQL-compatible database
driver = "file"
### The data source name (DSN) indicating the location of the checkpoint
→ storage.
### For the "file" driver, the DSN is a path. If the path is not specified,
→ TiDB Lightning would
### default to "/tmp/CHECKPOINT_SCHEMA.pb".
### For the "mysql" driver, the DSN is a URL in the form of "USER:PASS@tcp(
→ HOST:PORT)/".
### If the URL is not specified, the TiDB server from the [tidb] section is
→ used to
### store the checkpoints. You should specify a different MySQL-compatible
### database server to reduce the load of the target TiDB cluster.
### dsn = "/tmp/tidb_lightning_checkpoint.pb"
### Whether to keep the checkpoints after all data are imported. If false,
→ the
### checkpoints will be deleted. Keeping the checkpoints can aid debugging
→ but
### will leak metadata about the data source.
### keep-after-success = false

[tikv-importer]
### Delivery backend, can be "local", "importer" or "tidb".
### backend = "local"
### The listening address of tikv-importer when backend is "importer".
→ Change it to the actual address.
addr = "172.16.31.10:8287"
### Action to do when trying to insert a duplicated entry in the "tidb"
→ backend.
### - replace: use new entry to replace the existing entry
### - ignore: keep the existing entry, and ignore the new entry
### - error: report error and quit the program
### on-duplicate = "replace"
### state in the target TiDB.
### duplicate-resolution = 'none'
### The number of KV pairs sent in one request in the "local" backend.
### send-kv-pairs = 32768
### The directory of local KV sorting in the "local" backend. If the disk
### performance is low (such as in HDD), it is recommended to set the
→ directory
### on a different disk from `data-source-dir` to improve import speed.
### sorted-kv-dir = ""
### The concurrency that TiKV writes KV data in the "local" backend.
### When the network transmission speed between TiDB Lightning and TiKV

```

```

### exceeds 10 Gigabit, you can increase this value accordingly.
### range-concurrency = 16

[mydumper]
### Block size for file reading. Keep it longer than the longest string of
    ↪ the data source.
read-block-size = "64KiB" # default value

### The engine file needs to be imported sequentially. Due to parallel
    ↪ processing,
### multiple data engines will be imported at nearly the same time, and this
### creates a queue and wastes resources. Therefore, TiDB Lightning slightly
### increases the size of the first few batches to properly distribute
### resources. The scale up factor is controlled by this parameter, which
### expresses the ratio of duration between the "import" and "write" steps
### with full concurrency. This can be calculated by using the ratio
### (import duration/write duration) of a single table of size around 1 GiB.
### The exact timing can be found in the log. If "import" is faster, the
    ↪ batch
### size variance is smaller, and a ratio of zero means a uniform batch size
    ↪ .
### This value should be in the range (0 <= batch-import-ratio < 1).
batch-import-ratio = 0.75

### Local source data directory or the URL of the external storage.
data-source-dir = "/data/my_database"
### If no-schema is set to true, tidb-lightning assumes that the table
    ↪ skeletons
### already exist on the target TiDB cluster, and will not execute the ``
    ↪ CREATE
### TABLE` statements.
no-schema = false
### The character set of the schema files, containing CREATE TABLE
    ↪ statements;
### only supports one of:
### - utf8mb4: the schema files must be encoded as UTF-8; otherwise, an
    ↪ error is reported.
### - gb18030: the schema files must be encoded as GB-18030; otherwise,
    ↪ an error is reported
### - auto: (default) automatically detects whether the schema is UTF-8 or
    ↪ GB-18030. An error is reported if the encoding is neither.
### - binary: do not try to decode the schema files
character-set = "auto"

### Specifies the character set of the source data file. Lightning converts

```

```

    ↵ the source file from the specified character set to UTF-8 encoding
    ↵ when importing.

### Currently, this configuration only specifies the character set of the
    ↵ CSV files with the following options supported:
### - utf8mb4: Indicates that the source data file uses UTF-8 encoding.
### - GB18030: Indicates that the source data file uses the GB-18030
    ↵ encoding.
### - GBK: The source data file uses GBK encoding (GBK encoding is an
    ↵ extension of the GB-2312 character set, also known as Code Page 936).
### - binary: Indicates that Lightning does not convert the encoding (by
    ↵ default).
### If left blank, the default value "binary" is used, that is to say,
    ↵ Lightning does not convert the encoding.
### Note that Lightning does not predict about the character set of the
    ↵ source data file and only converts the source file and import the
    ↵ data based on this configuration.
### If the value of this configuration is not the same as the actual
    ↵ encoding of the source data file, a failed import, data loss or data
    ↵ disorder might appear.

data-character-set = "binary"
### Specifies the replacement character in case of incompatible characters
    ↵ during the character set conversion of the source data file.
### This configuration must not be duplicated with field separators, quote
    ↵ definers, and line breaks.
### The default value is "\uFFFD", which is the "error" Rune or Unicode
    ↵ replacement character in UTF-8 encoding.
### Changing the default value might result in potential degradation of
    ↵ parsing performance for the source data file.
data-invalid-char-replace = "\uFFFD"

### the input data in a "strict" format speeds up processing.
### "strict-format = true" requires that:
### in CSV, every value cannot contain literal new lines (U+000A and U+000D,
    ↵ or \r and \n) even
### when quoted, which means new lines are strictly used to separate rows.
### "Strict" format allows TiDB Lightning to quickly locate split positions
    ↵ of a large file for parallel processing.
### However, if the input data is not "strict", it may split a valid data in
    ↵ half and
### corrupt the result.
### The default value is false for safety instead of speed.
strict-format = false

### If strict-format is true, TiDB Lightning splits large CSV files into
    ↵ multiple chunks to process in

```

```

### parallel. max-region-size is the maximum size of each chunk after
    ↪ splitting.
### max-region-size = "256MiB" # default value

### Only import tables if these wildcard rules are matched. See the
    ↪ corresponding section for details.
filter = ['.*', '!mysql.*', '!sys.*', '!INFORMATION_SCHEMA.*', '!'
    ↪ PERFORMANCE_SCHEMA.*', '!METRICS_SCHEMA.*', '!INSPECTION_SCHEMA.*']

### Configures how CSV files are parsed.
[mydumper.csv]
### Separator between fields. Must not be empty.
separator = ','
### Quoting delimiter. Empty value means no quoting.
delimiter = ""
### Line terminator. Empty value means both "\n" (LF) and "\r\n" (CRLF) are
    ↪ line terminators.
terminator = ''
### Whether the CSV files contain a header.
### If `header` is true, the first line will be skipped.
header = true
### Whether the CSV contains any NULL value.
### If `not-null` is true, all columns from CSV cannot be NULL.
not-null = false
### When `not-null` is false (that is, CSV can contain NULL),
### fields equal to this value will be treated as NULL.
null = '\N'
### Whether to interpret backslash escapes inside fields.
backslash-escape = true
### If a line ends with a separator, remove it.
trim-last-separator = false

[tidb]
### Configuration of any TiDB server from the cluster.
host = "172.16.31.1"
port = 4000
user = "root"
password = ""
### Table schema information is fetched from TiDB via this status-port.
status-port = 10080
### Address of any PD server from the cluster.
pd-addr = "172.16.31.4:2379"
### tidb-lightning imports TiDB as a library and generates some logs itself.
### This setting controls the log level of the TiDB library.
log-level = "error"

```

```

### Sets the TiDB session variable to speed up the Checksum and Analyze
    → operations.
### See https://pingcap.com/docs/dev/reference/performance/statistics/# 
    → control-analyze-concurrency
### for the meaning of each setting
build-stats-concurrency = 20
distsql-scan-concurrency = 100
index-serial-scan-concurrency = 20
checksum-table-concurrency = 16

### The default SQL mode used to parse and execute the SQL statements.
sql-mode = "ONLY_FULL_GROUP_BY,NO_ENGINE_SUBSTITUTION"
### Sets maximum packet size allowed for SQL connections.
### Set this to 0 to automatically fetch the `max_allowed_packet` variable
    → from server on every connection.
max-allowed-packet = 67_108_864

### Whether to use TLS for SQL connections. Valid values are:
### * ""          - force TLS (same as "cluster") if [tidb.security]
    → section is populated, otherwise same as "false"
### * "false"     - disable TLS
### * "cluster"   - force TLS and verify the server's certificate with the
    → CA specified in the [tidb.security] section
### * "skip-verify" - force TLS but do not verify the server's certificate
    → (insecure!)
### * "preferred" - same as "skip-verify", but if the server does not
    → support TLS, fallback to unencrypted connection
### tls = ""

### Specifies certificates and keys for TLS-enabled MySQL connections.
### Defaults to a copy of the [security] section.
### [tidb.security]
### Public certificate of the CA. Set to empty string to disable TLS for SQL
    → .
### ca-path = "/path/to/ca.pem"
### Public certificate of this service. Default to copy of `security.cert-
    → path`
### cert-path = "/path/to/lightning.pem"
### Private key of this service. Default to copy of `security.key-path`
### key-path = "/path/to/lightning.key"

### When data importing is complete, tidb-lightning can automatically
    → perform
### the Checksum, Compact and Analyze operations. It is recommended to leave

```

```

### these as true in the production environment.
### The execution order: Checksum -> Analyze
[post-restore]
### Specifies whether to perform `ADMIN CHECKSUM TABLE <table>` for each
→ table to verify data integrity after importing.
### The following options are available:
### - "required" (default value): Perform admin checksum. If checksum fails,
→ TiDB Lightning will exit with failure.
### - "optional": Perform admin checksum. If checksum fails, TiDB Lightning
→ will report a WARN log but ignore any error.
### - "off": Do not perform checksum.
### Note that since v4.0.8, the default value has changed from "true" to "
→ required".
### For backward compatibility, bool values "true" and "false" are also
→ allowed for this field.
### "true" is equivalent to "required" and "false" is equivalent to "off".
checksum = "required"
### Specifies whether to perform `ANALYZE TABLE <table>` for each table
→ after checksum is done.
### Options available for this field are the same as `post-restore`. However
→ , the default value for this field is "optional".
analyze = "optional"

### If the value is set to `true`, a level-1 compaction is performed
### every time a table is imported.
### The default value is `false`.
level-1-compact = false

### If the value is set to `true`, a full compaction on the whole
### TiKV cluster is performed at the end of the import.
### The default value is `false`.
compact = false

### Configures the background periodic actions.
### Supported units: h (hour), m (minute), s (second).
[cron]
### Duration between which TiDB Lightning automatically refreshes the import
→ mode
### status. Should be shorter than the corresponding TiKV setting.
switch-mode = "5m"
### Duration between which an import progress is printed to the log.
log-progress = "5m"

```

### 10.8.5.1.3 TiKV Importer

```

### TiKV Importer configuration file template.

### Log file.
log-file = "tikv-importer.log"
### Log level: trace, debug, info, warn, error, off.
log-level = "info"

### Listening address of the status server. Prometheus can scrape metrics
    ↪ from this address.
status-server-address = "0.0.0.0:8286"

[server]
### The listening address of tikv-importer. tidb-lightning needs to connect
    ↪ to
### this address to write data.
addr = "0.0.0.0:8287"
### Size of the thread pool for the gRPC server.
grpc-concurrency = 16

[metric]
### These settings are relevant when using Prometheus Pushgateway. Normally
    ↪ you should let Prometheus
### to scrape metrics from the status-server-address.
### The Prometheus client push job name.
job = "tikv-importer"
### The Prometheus client push interval.
interval = "15s"
### The Prometheus Pushgateway address.
address = ""

[rocksdb]
### The maximum number of concurrent background jobs.
max-background-jobs = 32

[rocksdb.defaultcf]
### Amount of data to build up in memory before flushing data to the disk.
write-buffer-size = "1GB"
### The maximum number of write buffers that are built up in memory.
max-write-buffer-number = 8

### The compression algorithms used in different levels.
### The algorithm at level-0 is used to compress KV data.
### The algorithm at level-6 is used to compress SST files.
### The algorithms at level-1 to level-5 are unused for now.

```

```

compression-per-level = ["lz4", "no", "no", "no", "no", "no", "no", "lz4"]

[rocksdb.writecf]
### (same as above)
compression-per-level = ["lz4", "no", "no", "no", "no", "no", "no", "lz4"]

[security]
### The path for TLS certificates. Empty string means disabling secure
→ connections.
### ca-path = ""
### cert-path = ""
### key-path = ""

[import]
### The directory to store engine files.
import-dir = "/mnt/ssd/data.import/"
### Number of threads to handle RPC requests.
num-threads = 16
### Number of concurrent import jobs.
num-import-jobs = 24
### Maximum duration to prepare Regions.
#max-prepare-duration = "5m"
### Split Regions into this size according to the importing data.
#region-split-size = "512MB"
### Stream channel window size. The stream will be blocked on channel full.
#stream-channel-window = 128
### Maximum number of open engines.
max-open-engines = 8
### Maximum upload speed (bytes per second) from Importer to TiKV.
### upload-speed-limit = "512MB"
### Minimum ratio of available space on the target store:
→ store_available_space`/`store_capacity`.
### Importer pauses uploading SST if the availability ratio of the target
→ store is less than this
### value, to allow enough time for PD to balance Regions.
min-available-ratio = 0.05

```

### 10.8.5.2 Command line parameters

#### 10.8.5.2.1 Usage of `tidb-lightning`

Parameter	Explanation	Corresponding set-
<code>-</code>	Reads global configuration from <i>file</i> . If not specified, the default configuration would be used.	
<code>-V</code>	Prints program version	
<code>-d</code>	Directory	<code>mydumper</code>
<i>directory</i>	or <code>external</code>	$\hookrightarrow$ .
		<code>data</code>
	<code>storage</code>	$\hookrightarrow$ -
	<code>age</code>	$\hookrightarrow$ <code>source</code>
	<code>URL</code>	$\hookrightarrow$ -
	of the <code>data</code>	$\hookrightarrow$ <code>dir</code>
	dump to <code>read</code>	$\hookrightarrow$
	from	

---

Parameter	Explanation	Corresponding set-
<code>-L</code>	Log level: <code>lightning</code>	<code>log_level: &lt;-- .</code>
<i>level</i>	de- bug, info, warn, error, fatal (de- fault = info)	<code>log_level: &lt;-- -level</code>
<code>-f rule</code>	Table filter rules (can be specified multiple times)	<code>mydumper &lt;-- .filter &lt;-- filter (can_be_specified_multiple_times)</code>
<code>-</code>	Delivery backend end (local or tidb)	<code>tikv-&lt;-- importer &lt;-- .end &lt;-- local &lt;-- ,&lt;-- importer &lt;-- ,&lt;-- or &lt;-- tidb)</code>

---

Parameter	Explanation	Corresponding set-
<code>-log-</code>	Log	<code>lightning</code>
<code>file</code>	file	$\hookrightarrow$ .
<i>file</i>	path.	$\hookrightarrow$ <code>log</code>
	By de-	$\hookrightarrow$ -
	fault,	$\hookrightarrow$ <code>file</code>
	it is	$\hookrightarrow$
	<code>/tmp/</code>	
	$\hookrightarrow$ <code>lightning</code>	
	$\hookrightarrow$ .	
	$\hookrightarrow$ <code>log</code>	
	$\hookrightarrow$ .{	
	$\hookrightarrow$ <code>timestamp</code>	
	$\hookrightarrow$ }.	
	If set	
	to ‘-’,	
	it	
	means	
	that	
	the	
	log	
	files	
	will	
	be	
	out-	
	put to	
	std-	
	out.	
<code>-</code>	Listening	<code>lightning</code>
<code>status-</code>	ad-	$\hookrightarrow$ .
<code>addr</code>	dress	$\hookrightarrow$ <code>status</code>
<i>ip:port</i>	of the	$\hookrightarrow$ -
	TiDB	$\hookrightarrow$ <code>port</code>
	Light-	$\hookrightarrow$
	ning	
	server	
<code>-</code>	Address	<code>tikv-</code>
<code>importerof</code>		$\hookrightarrow$ <code>importer</code>
<i>host:port</i>	TiKV	$\hookrightarrow$ .
	Im-	$\hookrightarrow$ <code>addr</code>
	porter	$\hookrightarrow$

Parameter	Explanation	Corresponding set-
-pd-urls	PD end- <i>host:port</i>	tidb. → pd → - → addr → dress
-tidb-host	TiDB server <i>host</i>	tidb. → server → host
-tidb-port	TiDB server <i>port</i>	tidb. → server → port
(de-fault=4000)	port (de-fault fault = 4000)	port → port → = 4000)
-tidb-status	TiDB status <i>port</i>	tidb. → status → - → port = 10080)
-tidb-user	User name <i>user</i>	tidb. → user to connect to TiDB
-tidb-password	Password to <i>password</i>	tidb. → password to TiDB

---

Parameter	Explanation	Corresponding set-
<code>-no-schema</code>	Ignore schema files, get schema directly from TiDB.	<code>mydumper</code> → .
<code>-enable-checkpointable</code>	Whether checkpointable	→ <code>checkpointable</code>
<code>bool</code>	check-points (de-fault = true)	→ <code>enable</code>
<code>-analyze-tables</code>	Analyze post-analyze tables after import-ing.	→ <code>analyze</code>
<code>level</code>	Available values are “required”, “optional” (de-fault value), and “off”	→ <code>restore</code>
		→ .

---

Parameter	Explanation	Corresponding set-
–	Compare post-	
checksum	check- → <b>restore</b>	
<i>level</i>	sum → .	
	after → <b>checksum</b>	
	im- →	
	port- →	
	ing. →	
	Avail- →	
	able →	
	values →	
	are →	
	“re- →	
	quired” →	
	(de- →	
	fault →	
	value), →	
	“op- →	
	tional”, →	
	and →	
	“off” →	
–	Check → <b>lightning</b>	
check-	clus- → .	
	requirements → <b>check</b>	
<i>bool</i>	ver- → -	
	sion → <b>requirements</b>	
	com- →	
	pati- →	
	bility →	
	before →	
	start- →	
	ing →	
	(de- →	
	fault →	
	= →	
	true) →	

Parameter	Explanation	Corresponding setting
<code>-ca file</code>	CA certificate path for TLS connection	<code>security.ca</code>
<code>-cert file</code>	Certificate path for TLS connection	<code>security.cert</code>
<code>-key file</code>	Private key path for TLS connection	<code>security.key</code>
<code>-server-mode</code>	Start TiDB in Lightningning server mode	<code>lightning.server</code>

If a command line parameter and the corresponding setting in the configuration file are both provided, the command line parameter will be used. For example, running `./tidb -lightning -L debug --config cfg.toml` would always set the log level to “debug” regardless of the content of `cfg.toml`.

#### 10.8.5.3 Usage of `tidb-lightning-ctl`

This tool can execute various actions given one of the following parameters:

Parameter	Explanation
-c	Performs compact a full compaction
-S	Switches switch-mode every TiKV mode
store	store to the given mode: normal, import
-fetch-mode	Prints the current mode of every TiKV store
-import-engine	Imports the closed engine
uuid	engine file from TiKV Importer into the TiKV cluster

Parameter	Explanation
-	Deletes
cleanup-	the
engine	engine
<i>uuid</i>	file
	from
	TiKV
Im-	
porter	
-	Dumps
checkpoint	the
dump	rent
<i>folder</i>	check-
	point
	as
	CSVs
	into
	the
	folder
-	Removes
checkpoint	the
error-	check-
destroy	point
<i>table-</i>	and
<i>name</i>	drops
	the
	table
	if it
	caused
	error
-	Ignores
checkpoint	the
error-	error
ignore	recorded
<i>table-</i>	in the
<i>name</i>	check-
	point
	in-
	volv-
	ing
	the
	given
	table

Parameter	Explanation
-C, --config <i>file</i>	Unconditionally removes the check-point of the table.

The *tablename* must either be a qualified table name in the form `db`.`tbl` (including the backquotes), or the keyword “all”.

Additionally, all parameters of `tidb-lightning` described in the section above are valid in `tidb-lightning-ctl`.

#### 10.8.5.4 Usage of `tikv-importer`

Parameter	Corresponding setting
-C, --config <i>file</i>	Reads configuration from <i>file</i> . If not specified, the default configuration would be used.
-V, --version	Prints program version

Parameter	Explanation	Corresponding set-
-A, <i>ip:port</i>	Listening server address of the TiKV Im-porter server	server
-status- <i>ip:port</i>	Listening status of the server	status
-import- <i>dir</i>	Stores files in this directory	import
-log-level- <i>level</i>	Log level: trace, de-bug, info, warn, error, off	log-level
-log-file- <i>file</i>	Log file path	log-file

## 10.8.6 Key Features

### 10.8.6.1 TiDB Lightning Checkpoints

Importing a large database usually takes hours or days, and if such long running processes spuriously crashes, it can be very time-wasting to redo the previously completed tasks. To solve this, TiDB Lightning uses *checkpoints* to store the import progress, so that `tidb-lightning` continues importing from where it lefts off after restarting.

This document describes how to enable, configure, store, and control *checkpoints*.

#### 10.8.6.1.1 Enable and configure checkpoints

```
[checkpoint]
##### Whether to enable checkpoints.
##### While importing data, TiDB Lightning records which tables have been
    ↪ imported, so
##### even if TiDB Lightning or some other component crashes, you can start
    ↪ from a known
##### good state instead of restarting from scratch.
enable = true

##### Where to store the checkpoints.
##### - file: store as a local file (requires v2.1.1 or later)
##### - mysql: store into a remote MySQL-compatible database
driver = "file"

##### The schema name (database name) to store the checkpoints
##### Enabled only when `driver = "mysql"`.
##### schema = "tidb_lightning_checkpoint"

##### The data source name (DSN) indicating the location of the checkpoint
    ↪ storage.
##### # For the "file" driver, the DSN is a path. If the path is not
    ↪ specified, Lightning would
##### default to "/tmp/CHECKPOINT_SCHEMA.pb".
##### # For the "mysql" driver, the DSN is a URL in the form of "USER:
    ↪ PASS@tcp(HOST:PORT)/".
##### If the URL is not specified, the TiDB server from the [tidb] section is
    ↪ used to
##### store the checkpoints. You should specify a different MySQL-compatible
##### database server to reduce the load of the target TiDB cluster.
#dsn = "/tmp/tidb_lightning_checkpoint.pb"

##### Whether to keep the checkpoints after all data are imported. If false,
    ↪ the
##### checkpoints are deleted. Keeping the checkpoints can aid debugging but
##### might leak metadata about the data source.
##### keep-after-success = false
```

#### 10.8.6.1.2 Checkpoints storage

TiDB Lightning supports two kinds of checkpoint storage: a local file or a remote MySQL-compatible database.

- With `driver = "file"`, checkpoints are stored in a local file at the path given by the `dsn` setting. Checkpoints are updated rapidly, so we highly recommend placing the checkpoint file on a drive with very high write endurance, such as a RAM disk.
- With `driver = "mysql"`, checkpoints can be saved in any databases compatible with MySQL 5.7 or later, including MariaDB and TiDB. By default, the checkpoints are saved in the target database.

While using the target database as the checkpoints storage, Lightning is importing large amounts of data at the same time. This puts extra stress on the target database and sometimes leads to communication timeout. Therefore, **it is strongly recommended to install a temporary MySQL server to store these checkpoints**. This server can be installed on the same host as `tidb-lightning` and can be uninstalled after the importer progress is completed.

#### 10.8.6.1.3 Checkpoints control

If `tidb-lightning` exits abnormally due to unrecoverable errors (for example, data corruption), it refuses to reuse the checkpoints until the errors are resolved. This is to prevent worsening the situation. The checkpoint errors can be resolved using the `tidb-lightning-ctl` program.

```
--checkpoint-error-destroy
```

```
tidb-lightning-ctl --checkpoint-error-destroy='`schema`.`table`'
```

This option allows you to restart importing the table from scratch. The schema and table names must be quoted with backquotes and are case-sensitive.

- If importing the table ``schema`.`table`` failed previously, this option executes the following operations:
  - DROPs the table ``schema`.`table`` from the target database, which means removing all imported data.
  - Resets the checkpoints record of this table to be “not yet started”.
- If there is no errors involving the table ``schema`.`table``, this operation does nothing.

It is the same as applying the above on every table. This is the most convenient, safe and conservative solution to fix the checkpoint error problem:

```
tidb-lightning-ctl --checkpoint-error-destroy=all
```

```
--checkpoint-error-ignore
```

```
tidb-lightning-ctl --checkpoint-error-ignore='`schema`.`table`'
tidb-lightning-ctl --checkpoint-error-ignore=all
```

If importing the table `schema`.`table` failed previously, this clears the error status as if nothing ever happened. The `all` variant applies this operation to all tables.

#### Note:

Use this option only when you are sure that the error can indeed be ignored. If not, some imported data can be lost. The only safety net is the final “checksum” check, and thus you need to keep the “checksum” option always enabled when using `--checkpoint-error-ignore`.

`--checkpoint-remove`

```
tidb-lightning-ctl --checkpoint-remove='`schema`.`table`'
tidb-lightning-ctl --checkpoint-remove=all
```

This option simply removes all checkpoint information about one table or all tables, regardless of their status.

`--checkpoint-dump`

```
tidb-lightning-ctl --checkpoint-dump=output/directory
```

This option dumps the content of the checkpoint into the given directory, which is mainly used for debugging by the technical staff. This option is only enabled when `driver = "mysql"` ↫ ".

### 10.8.6.2 Table Filter

The TiDB ecosystem tools operate on all the databases by default, but oftentimes only a subset is needed. For example, you only want to work with the schemas in the form of `foo*` and `bar*` and nothing else.

Since TiDB 4.0, all TiDB ecosystem tools share a common filter syntax to define subsets. This document describes how to use the table filter feature.

#### 10.8.6.2.1 Usage

##### CLI

Table filters can be applied to the tools using multiple `-f` or `--filter` command line parameters. Each filter is in the form of `db.table`, where each part can be a wildcard (further explained in the [next section](#)). The following lists the example usage in each tool.

- `BR`:

```
./br backup full -f 'foo.*.*' -f 'bar.*.*' -s 'local:///tmp/backup'
#
#          ^~~~~~
```

```
./br restore full -f 'foo.*.*' -f 'bar.*.*' -s 'local:///tmp/backup'
#
#          ^~~~~~
```

- Dumpling:

```
./dumpling -f 'foo.*.*' -f 'bar.*.*' -P 3306 -o /tmp/data/
#
#          ^~~~~~
```

- TiDB Lightning:

```
./tidb-lightning -f 'foo.*.*' -f 'bar.*.*' -d /tmp/data/ --backend tidb
#
#          ^~~~~~
```

TOML configuration files

Table filters in TOML files are specified as [array of strings](#). The following lists the example usage in each tool.

- TiDB Lightning:

```
[mydumper]
filter = ['foo.*.*', 'bar.*.*']
```

- TiCDC:

```
[filter]
rules = ['foo.*.*', 'bar.*.*']

[[sink.dispatchers]]
matcher = ['db1.*', 'db2.*', 'db3.*']
dispatcher = 'ts'
```

### 10.8.6.2.2 Syntax

Plain table names

Each table filter rule consists of a “schema pattern” and a “table pattern”, separated by a dot (.). Tables whose fully-qualified name matches the rules are accepted.

```
db1.tbl1
db2.tbl2
db3.tbl3
```

A plain name must only consist of valid [identifier characters](#), such as:

- digits (0 to 9)
- letters (a to z, A to Z)
- \$
- \_
- non ASCII characters (U+0080 to U+10FFFF)

All other ASCII characters are reserved. Some punctuations have special meanings, as described in the next section.

### Wildcards

Each part of the name can be a wildcard symbol described in [fnmatch\(3\)](#):

- \* — matches zero or more characters
- ? — matches one character
- [a-z] — matches one character between “a” and “z” inclusively
- [!a-z] — matches one character except “a” to “z”.

```
db[0-9].tbl[0-9a-f][0-9a-f]
data.*
*.backup_*
```

“Character” here means a Unicode code point, such as:

- U+00E9 (é) is 1 character.
- U+0065 U+0301 (é) are 2 characters.
- U+1F926 U+1F3FF U+200D U+2640 U+FE0F ( ) are 5 characters.

### File import

To import a file as the filter rule, include an @ at the beginning of the rule to specify the file name. The table filter parser treats each line of the imported file as additional filter rules.

For example, if a file `config/filter.txt` has the following content:

```
employees.*
*.WorkOrder
```

the following two invocations are equivalent:

```
./dumpling -f '@config/filter.txt'
./dumpling -f 'employees.*' -f '*.WorkOrder'
```

A filter file cannot further import another file.

### Comments and blank lines

Inside a filter file, leading and trailing white-spaces of every line are trimmed. Furthermore, blank lines (empty strings) are ignored.

A leading # marks a comment and is ignored. # not at start of line is considered syntax error.

```
#### this line is a comment
db.table # but this part is not comment and may cause error
```

### Exclusion

An ! at the beginning of the rule means the pattern after it is used to exclude tables from being processed. This effectively turns the filter into a block list.

```
*.*
#^ note: must add the *.* to include all tables first
!*>Password
!employees.salaries
```

### Escape character

To turn a special character into an identifier character, precede it with a backslash \.

```
db\.with\.\dots.*
```

For simplicity and future compatibility, the following sequences are prohibited:

- \ at the end of the line after trimming whitespaces (use [ ] to match a literal whitespace at the end).
- \ followed by any ASCII alphanumeric character ([0-9a-zA-Z]). In particular, C-like escape sequences like \0, \r, \n and \t currently are meaningless.

### Quoted identifier

Besides \, special characters can also be suppressed by quoting using " or `.

```
"db.with.dots"."tbl\1"
`db.with.dots`.`tbl\2`
```

The quotation mark can be included within an identifier by doubling itself.

```
"foo""bar".`foo``bar`
#### equivalent to:
foo\"bar.foo`\bar
```

Quoted identifiers cannot span multiple lines.

It is invalid to partially quote an identifier:

```
"this is "invalid*.*
```

## Regular expression

In case very complex rules are needed, each pattern can be written as a regular expression delimited with /:

```
/^db\d{2,}$/ ./^tbl\d{2,}$/
```

These regular expressions use the [Go dialect](#). The pattern is matched if the identifier contains a substring matching the regular expression. For instance, /b/ matches db01.

### Note:

Every / in the regular expression must be escaped as \/, including inside [...] . You cannot place an unescaped / between \Q...\\E.

### 10.8.6.2.3 Multiple rules

When a table name matches none of the rules in the filter list, the default behavior is to ignore such unmatched tables.

To build a block list, an explicit \*.\* must be used as the first rule, otherwise all tables will be excluded.

```
#### every table will be filtered out
./dumpling -f '!*.Password'

#### only the "Password" table is filtered out, the rest are included.
./dumpling -f '*.*' -f '!*.Password'
```

In a filter list, if a table name matches multiple patterns, the last match decides the outcome. For instance:

```
#### rule 1
employees.*
#### rule 2
!* .dep*
#### rule 3
*.departments
```

The filtered outcome is as follows:

Table name	Rule 1	Rule 2	Rule 3	Outcome
irrelevant.table				Default (reject)
employees.employees				Rule 1 (accept)
employees.dept_emp				Rule 2 (reject)

Table name	Rule 1	Rule 2	Rule 3	Outcome
employees.departments				Rule 3 (accept)
else.departments				Rule 3 (accept)

#### Note:

In TiDB tools, the system schemas are always excluded in the default configuration. The system schemas are:

- INFORMATION\_SCHEMA
- PERFORMANCE\_SCHEMA
- METRICS\_SCHEMA
- INSPECTION\_SCHEMA
- mysql
- sys

### 10.8.6.3 TiDB Lightning CSV Support and Restrictions

This document describes how to migrate data from CSV files to TiDB using TiDB Lightning. For information about how to generate CSV files from MySQL, see [Export to CSV files using Dumpling](#).

TiDB Lightning supports reading CSV (comma-separated values) data source, as well as other delimited format such as TSV (tab-separated values).

#### 10.8.6.3.1 File name

A CSV file representing a whole table must be named as `db_name.table_name.csv`. This will be restored as a table `table_name` inside the database `db_name`.

If a table spans multiple CSV files, they should be named like `db_name.table_name → .003.csv`. The number part do not need to be continuous, but must be increasing and zero-padded.

The file extension must be `*.csv`, even if the content is not separated by commas.

#### 10.8.6.3.2 Schema

CSV files are schema-less. To import them into TiDB, a table schema must be provided. This could be done either by:

- Providing a file named `db_name.table_name-schema.sql` containing the `CREATE TABLE` DDL statement, and also a file named `db_name-schema-create.sql` containing the `CREATE DATABASE` DDL statement.

- Creating the empty tables directly in TiDB in the first place, and then setting [  
↪ mydumper] no-schema = true in `tidb-lightning.toml`.

#### 10.8.6.3.3 Configuration

The CSV format can be configured in `tidb-lightning.toml` under the [mydumper.csv] section. Most settings have a corresponding option in the MySQL `LOAD DATA` statement.

```
[mydumper.csv]
#### Separator between fields. Must be ASCII characters. It is not
    ↪ recommended to use the default ',',. It is recommended to use '\|+\|'
    ↪ or other uncommon character combinations.
separator = ','
#### Quoting delimiter. Empty value means no quoting.
delimiter = ""
#### Line terminator. Empty value means both "\n" (LF) and "\r\n" (CRLF) are
    ↪ line terminators.
terminator = ''
#### Whether the CSV files contain a header.
#### If `header` is true, the first line will be skipped.
header = true
#### Whether the CSV contains any NULL value.
#### If `not-null` is true, all columns from CSV cannot be NULL.
not-null = false
#### When `not-null` is false (that is, CSV can contain NULL),
#### fields equal to this value will be treated as NULL.
null = '\N'
#### Whether to interpret backslash escapes inside fields.
backslash-escape = true
#### If a line ends with a separator, remove it.
trim-last-separator = false
```

In all string fields such as `separator`, `delimiter` and `terminator`, if the input involves special characters, you can use backslash escape sequence to represent them in a *double-quoted* string ("..."). For example, `separator = "\u001f"` means using the ASCII character 0x1F as separator.

Additionally, you can use *single-quoted* strings ('...') to suppress backslash escaping. For example, `terminator = '\n'` means using the two-character string: a backslash followed by the letter "n", as the terminator.

See the [TOML v1.0.0 specification](#) for details.

`separator`

- Defines the field separator.
- Can be multiple characters, but must not be empty.

- Common values:
  - `' , '` for CSV (comma-separated values)
  - `"\t"` for TSV (tab-separated values)
  - `"\u0001"` to use the ASCII character 0x01 as separator
- Corresponds to the `FIELDS TERMINATED BY` option in the `LOAD DATA` statement.

#### `delimiter`

- Defines the delimiter used for quoting.
- If `delimiter` is empty, all fields are unquoted.
- Common values:
  - `''''` quote fields with double-quote, same as [RFC 4180](#)
  - `''` disable quoting
- Corresponds to the `FIELDS ENCLOSED BY` option in the `LOAD DATA` statement.

#### `terminator`

- Defines the line terminator.
- If `terminator` is empty, both `"\r"` (U+000D Carriage Return) and `"\n"` (U+000A Line Feed) are used as terminator.
- Corresponds to the `LINES TERMINATED BY` option in the `LOAD DATA` statement.

#### `header`

- Whether *all* CSV files contain a header row.
- If `header` is true, the first row will be used as the *column names*. If `header` is false, the first row is not special and treated as an ordinary data row.

#### `not-null` and `null`

- The `not-null` setting controls whether all fields are non-nullable.
- If `not-null` is false, the string specified by `null` will be transformed to the SQL `NUL` instead of a concrete value.
- Quoting will not affect whether a field is null.

For example, with the CSV file:

A,B,C \N,"\\N",
--------------------

In the default settings (`not-null = false; null = '\N'`), the columns A and B are both converted to NULL after importing to TiDB. The column C is simply the empty string '' but not NULL.

#### `backslash-escape`

- Whether to interpret backslash escapes inside fields.
- If `backslash-escape` is true, the following sequences are recognized and transformed:

Sequence	Converted to
\0	Null character (U+0000)
\b	Backspace (U+0008)
\n	Line feed (U+000A)
\r	Carriage return (U+000D)
\t	Tab (U+0009)
\Z	Windows EOF (U+001A)

In all other cases (for example, \") the backslash is simply stripped, leaving the next character (") in the field. The character left has no special roles (for example, delimiters) and is just an ordinary character.

- Quoting will not affect whether backslash escapes are interpreted.
- Corresponds to the `FIELDS ESCAPED BY '\'` option in the `LOAD DATA` statement.

#### `trim-last-separator`

- Treats the field `separator` as a terminator, and removes all trailing separators.

For example, with the CSV file:

A,,B,,
--------

- When `trim-last-separator = false`, this is interpreted as a row of 5 fields ('A', → '', 'B', '', '').
- When `trim-last-separator = true`, this is interpreted as a row of 3 fields ('A', → '', 'B').
- This option is deprecated, because the behavior with multiple trailing separators is not intuitive. Use the `terminator` option instead. If your old configuration was

```
separator = ',',  
trim-last-separator = true
```

we recommend changing this to

```
separator = ',',  
terminator = ",\n"
```

### Non-configurable options

TiDB Lightning does not support every option supported by the `LOAD DATA` statement. Some examples:

- There cannot be line prefixes (`LINES STARTING BY`).
- The header cannot be simply skipped (`IGNORE n LINES`). It must be valid column names if present.

#### 10.8.6.3.4 Strict format

Lightning works the best when the input files have uniform size around 256 MB. When the input is a single huge CSV file, Lightning can only use one thread to process it, which slows down import speed a lot.

This can be fixed by splitting the CSV into multiple files first. For the generic CSV format, there is no way to quickly identify when a row starts and ends without reading the whole file. Therefore, Lightning by default does *not* automatically split a CSV file. However, if you are certain that the CSV input adheres to certain restrictions, you can enable the `strict-format` setting to allow Lightning to split the file into multiple 256 MB-sized chunks for parallel processing.

```
[mydumper]  
strict-format = true
```

Currently, a strict CSV file means every field occupies only a single line. In other words, one of the following must be true:

- Delimiter is empty, or
- Every field does not contain the terminator itself. In the default configuration, this means every field does not contain CR (\r) or LF (\n).

If a CSV file is not strict, but `strict-format` was wrongly set to `true`, a field spanning multiple lines may be cut in half into two chunks, causing parse failure, or even worse, quietly importing corrupted data.

#### 10.8.6.3.5 Common configurations

##### CSV

The default setting is already tuned for CSV following RFC 4180.

```
[mydumper.csv]
separator = ',' # It is not recommended to use the default ',' . It is
    ↪ recommended to use '\|+\| ' or other uncommon character combinations
    ↪ .
delimiter = ''
header = true
not-null = false
null = '\N'
backslash-escape = true
```

Example content:

```
ID,Region,Count
1,"East",32
2,"South",\N
3,"West",10
4,"North",39
```

##### TSV

```
[mydumper.csv]
separator = "\t"
delimiter = ''
header = true
not-null = false
null = 'NULL'
backslash-escape = false
```

Example content:

ID	Region	Count
1	East	32
2	South	NULL
3	West	10
4	North	39

##### TPC-H DBGEN

```
[mydumper.csv]
separator = '|'
delimiter = ''
terminator = "|\\n"
header = false
```

```
not-null = true
backslash-escape = false
```

Example content:

```
1|East|32|
2|South|0|
3|West|10|
4|North|39|
```

#### 10.8.6.4 TiDB Lightning Backends

The backend determines how TiDB Lightning imports data into the target cluster.

TiDB Lightning supports the following [backends](#):

- [Local-backend](#)
- [Importer-backend](#)
- [TiDB-backend](#)

The **Local-backend**: `tidb-lightning` first encodes data into key-value pairs, sorts and stores them in a local temporary directory, and *upload* these key-value pairs to each TiKV node as *SST files*. Then, TiKV ingests these *SST files* into the cluster. The implementation of Local-backend is the same with that of Importer-backend but does not rely on the external `tikv-importer` component.

The **Importer-backend**: `tidb-lightning` first encodes the SQL or CSV data into KV pairs, and relies on the external `tikv-importer` program to sort these KV pairs and ingest directly into the TiKV nodes.

The **TiDB-backend**: `tidb-lightning` first encodes these data into SQL `INSERT` statements, and has these statements executed directly on the TiDB node.

Backend	Local-backend	Importer-backend	TiDB-backend
Speed	Fast (~500 GB/hr)	Fast (~300 GB/hr)	Slow (~50 GB/hr)
Resource usage	High	High	Low
Network bandwidth usage	High	Medium	Low
ACID respected while importing	No	No	Yes
Target tables	Must be empty	Must be empty	Can be populated
Additional component required	No	<code>tikv-importer</code>	No
TiDB versions supported	<code>&gt;= v4.0.0</code>	All	All
TiDB services impacted	Yes	Yes	No

##### 10.8.6.4.1 How to choose the backend modes

- If the target cluster of data import is v4.0 or later versions, consider using the Local-backend mode first, which is easier to use and has higher performance than that of the other two modes.
- If the target cluster of data import is v3.x or earlier versions, it is recommended to use the Importer-backend mode.
- If the target cluster of data import is in the online production environment, or if the target table of data import already has data on it, it is recommended to use the TiDB-backend mode.

#### 10.8.6.4.2 TiDB Lightning Local-backend

The Local-backend feature is introduced to TiDB Lightning since TiDB v4.0.3. You can use this feature to import data to TiDB clusters of v4.0.0 or above.

Deployment for Local-backend

To deploy TiDB Lightning in the Local-backend mode, see [TiDB Lightning Deployment](#).

#### 10.8.6.4.3 TiDB Lightning TiDB-backend

##### Note:

Since TiDB v4.0, PingCAP no longer maintains the [Loader](#) tool. Since v5.0, the Loader documentation is no longer available. Loader's functionality has been completely replaced by the TiDB-backend of TiDB Lightning, so it is highly recommended to switch to TiDB Lightning.

Deployment for TiDB-backend

When using the TiDB-backend, deploying `tikv-importer` is not necessary. Compared with the [standard deployment procedure](#), the TiDB-backend deployment has the following two differences:

- All steps involving `tikv-importer` can be skipped.
- The configuration must be changed to declare that the TiDB-backend is used.

Hardware requirements

The speed of TiDB Lightning using TiDB-backend is limited by the SQL processing speed of TiDB. Therefore, even a lower-end machine may max out the possible performance. The recommended hardware configuration is:

- 16 logical cores CPU

- An SSD large enough to store the entire data source, preferring higher read speed
- 1 Gigabit network card

### Manual deployment

You do not need to download and configure `tikv-importer`. You can download TiDB Lightning from [here](#).

Before running `tidb-lightning`, add the following lines into the configuration file:

```
[tikv-importer]
backend = "tidb"
```

or supplying the `--backend tidb` arguments when executing `tidb-lightning`.

Configuration description and samples

This section provides the samples for task configuration in TiDB Lightning.

```
#### tidb-lightning task configuration

[lightning]
#### Checks whether the cluster satisfies the minimum requirement before
    ↪ starting.
check-requirements = true

#### Each table is split into one "index engine" to store indices, and
    ↪ multiple
#### "data engines" to store row data. These settings control the maximum
    #### concurrent number for each type of engines.
#### Controls the maximum number of tables that can be imported in parallel.
    ↪ For TiDB-backend, the default value is the number of CPU cores.
index-concurrency = 40

#### Controls the maximum number of "data engines" allowed to be imported in
    ↪ parallel. The default value is the number of CPU cores. The value
    ↪ should be no less than the value of index-concurrency.
table-concurrency = 40

#### The number of concurrent SQL statements executed. It is set to the
    ↪ number of logical CPU cores by default. The bottleneck of TiDB-
    ↪ backend is usually not the CPU. You can increase this value based on
    ↪ the actual load of the downstream cluster to optimize the write speed
    ↪ . At the same time, when adjusting this configuration, it is
    ↪ recommended to adjust the index-concurrency and table-concurrency to
    ↪ the same value.
region-concurrency = 40
```

```

##### Logging
level = "info"

##### The directory to which the log is output. If it is empty (default), the
    ↪ file is saved to /tmp/lightning.log.{timestamp}. If you want the
    ↪ logs to be written to the system standard output, set it to "-".
file = "tidb-lightning.log"

[checkpoint]
##### Whether to enable checkpoints.
##### While importing data, TiDB Lightning records which tables have been
    ↪ imported, so
##### even if TiDB Lightning or some other component crashes, you can start
    ↪ from a known
##### good state instead of restarting from scratch.
enable = true

##### Where to store the checkpoints.
##### - file (default): store as a local file (requires v2.1.1 or later)
##### - mysql: store into a remote MySQL-compatible database
driver = "file"

##### The schema name (database name) to store the checkpoints
##### Enabled only when `driver = "mysql"`.
##### schema = "tidb_lightning_checkpoint"

##### The data source name (DSN) indicating the location of the checkpoint
    ↪ storage.
##### # For the "file" driver, the DSN is a path. If the path is not
    ↪ specified, Lightning would
##### default to "/tmp/CHECKPOINT_SCHEMA.pb".
##### # For the "mysql" driver, the DSN is a URL in the form of "USER:
    ↪ PASS@tcp(HOST:PORT)/".
##### If the URL is not specified, the TiDB server from the [tidb] section is
    ↪ used to
##### store the checkpoints. You should specify a different MySQL-compatible
##### database server to reduce the load of the target TiDB cluster.
#dsn = "/tmp/tidb_lightning_checkpoint.pb"

##### Whether to keep the checkpoints after all data are imported. If false,
    ↪ the
##### checkpoints are deleted. Keeping the checkpoints can aid debugging but
##### might leak metadata about the data source.
##### keep-after-success = false

[tikv-importer]

```

```

##### use the TiDB-backend.
backend = "tidb"

##### Action to do when trying to insert a duplicated entry in the "tidb"
    ↪ backend.
##### - replace: use new entry to replace the existing entry
##### - ignore: keep the existing entry, and ignore the new entry
##### - error: report error and quit the program
##### on-duplicate = "replace"

[mydumper]
##### Block size for file reading. Keep it longer than the longest string of
##### the data source.
##### read-block-size = "64KiB"

##### Minimum size (in terms of source data file) of each batch of import.
##### TiDB Lightning splits a large table into multiple data engine files
    ↪ according to this size.
##### batch-size = 107_374_182_400 # Byte (default = 100 GB)

##### Local source data directory or the URL of the external storage.
data-source-dir = "/data/my_database"

##### the input data in a "strict" format speeds up processing.
##### "strict-format = true" requires that:
##### in CSV, every value cannot contain literal new lines (U+000A and U+000D
    ↪ , or \r and \n) even
##### when quoted, which means new lines are strictly used to separate rows.
##### "Strict" format allows TiDB Lightning to quickly locate split positions
    ↪ of a large file for parallel processing.
##### However, if the input data is not "strict", it may split a valid data
    ↪ in half and
##### corrupt the result.
##### The default value is false for safety instead of speed.
strict-format = false

##### If strict-format is true, TiDB Lightning splits large CSV files into
    ↪ multiple chunks to process in
##### parallel. max-region-size is the maximum size of each chunk after
    ↪ splitting.
##### max-region-size = 268_435_456 # Byte (default = 256 MB)

##### Only import tables if these wildcard rules are matched. See the
    ↪ corresponding section for details.
filter = ['.*', '!mysql.*', '!sys.*', '!INFORMATION_SCHEMA.*', '!'

```

```

↪ PERFORMANCE_SCHEMA.*', '!METRICS_SCHEMA.*', '!INSPECTION_SCHEMA.*']

#### Configures how CSV files are parsed.
[mydumper.csv]
#### Separator between fields, should be an ASCII character.
separator = ','
#### Quoting delimiter, can either be an ASCII character or empty string.
delimiter = ''
#### Whether the CSV files contain a header.
#### If `header` is true, the first line will be skipped.
header = true
#### Whether the CSV contains any NULL value.
#### If `not-null` is true, all columns from CSV cannot be NULL.
not-null = false
#### When `not-null` is false (that is, CSV can contain NULL),
#### fields equal to this value will be treated as NULL.
null = '\N'
#### Whether to interpret backslash escapes inside fields.
backslash-escape = true
#### If a line ends with a separator, remove it.
trim-last-separator = false

[tidb]
#### Configuration of any TiDB server from the cluster.
host = "172.16.31.1"
port = 4000
user = "root"
password = ""

#### The default SQL mode used to parse and execute the SQL statements.
sql-mode = "ONLY_FULL_GROUP_BY,NO_ENGINE_SUBSTITUTION"

#### Whether to use TLS for SQL connections. Valid values are:
#### * "" - force TLS (same as "cluster") if [tidb.security]
    ↪ section is populated, otherwise same as "false"
#### * "false" - disable TLS
#### * "cluster" - force TLS and verify the server's certificate with the
    ↪ CA specified in the [tidb.security] section
#### * "skip-verify" - force TLS but do not verify the server's certificate
    ↪ (insecure!)
#### * "preferred" - same as "skip-verify", but if the server does not
    ↪ support TLS, fallback to unencrypted connection
#### tls = ""

#### Specifies certificates and keys for TLS-enabled MySQL connections.

```

```
##### [tidb.security]

##### Public certificate of the CA. Set to empty string to disable TLS for
    ↪ SQL.
##### ca-path = "/path/to/ca.pem"

##### Public certificate of this service. Default to copy of `security.cert-
    ↪ path`
##### cert-path = "/path/to/lightning.pem"

##### Private key of this service. Default to copy of `security.key-path`
##### key-path = "/path/to/lightning.key"

##### Configures the background periodic actions.
##### Supported units: h (hour), m (minute), s (second).
[cron]

##### Duration between which an import progress is printed to the log.
log-progress = "5m"
```

For detailed descriptions of the configuration items, see [TiDB Lightning Configuration](#).

#### Conflict resolution

The TiDB-backend supports importing to an already-populated table. However, the new data might cause a unique key conflict with the old data. You can control how to resolve the conflict by using this task configuration.

```
[tikv-importer]
backend = "tidb"
on-duplicate = "replace" # or "error" or "ignore"
```

Setting	Behavior on conflict	Equivalent SQL statement
replace	New entries replace old ones	REPLACE INTO ...
ignore	Keep old entries and ignore new ones	INSERT IGNORE INTO ...
error	Abort import	INSERT INTO ...

#### Migrating from Loader to TiDB Lightning TiDB-backend

If you need to import data into a TiDB cluster, TiDB Lightning using the TiDB-backend can completely replace the functionalities of [Loader](#). The following list shows how to translate Loader configurations into [TiDB Lightning configurations](#).

Loader

TiDB Lightning

```
#### log level
log-level = "info"

#### The directory to which the log is output
log-file = "loader.log"

#### Prometheus
status-addr = ":8272"

#### concurrency
pool-size = 16
```

```
[lightning]
#### log level
level = "info"

#### The directory to which the log is output. If this directory is not
    ↪ specified, it defaults to the directory where the command is executed
    ↪ .
file = "tidb-lightning.log"

#### Prometheus
pprof-port = 8289

#### concurrency (better left as default)
#region-concurrency = 16
```

```
#### checkpoint database
checkpoint-schema = "tidb_loader"
```

```
[checkpoint]
#### checkpoint storage
enable = true
schema = "tidb_lightning_checkpoint"
#### by default the checkpoint is stored in
#### a local file, which is more efficient.
#### but you could still choose to store the
#### checkpoints in the target database with
#### this setting:
#driver = "mysql"
```

```
[tikv-importer]
##### use the TiDB-backend
backend = "tidb"
```

```
##### data source directory
dir = "/data/export/"
```

```
[mydumper]
##### data source directory
data-source-dir = "/data/export"
```

```
[db]
##### TiDB connection parameters
host = "127.0.0.1"
port = 4000

user = "root"
password = ""

#sql-mode = ""
```

```
[tidb]
##### TiDB connection parameters
host = "127.0.0.1"
port = 4000

##### In the TiDB-backend mode, this parameter is optional.
##### status-port = 10080
user = "root"
password = ""

#sql-mode = ""
```

```
#####[[route-rules]]
##### Table routes
##### schema-pattern = "shard_db_"
##### table-pattern = "shard_table_"
##### target-schema = "shard_db"
##### target-table = "shard_table"
```

```
#####[[routes]]
##### schema-pattern = "shard_db_"
##### table-pattern = "shard_table_"
```

```
#### target-schema = "shard_db"
#### target-table = "shard_table"
```

#### 10.8.6.4.4 TiDB Lightning Importer-backend

Deployment for Importer-backend mode

This section describes how to [deploy TiDB Lightning manually](#) in the Importer-backend mode:

Hardware requirements

`tidb-lightning` and `tikv-importer` are both resource-intensive programs. It is recommended to deploy them into two separate machines.

To achieve the best performance, it is recommended to use the following hardware configuration:

- **`tidb-lightning`:**
  - 32+ logical cores CPU
  - An SSD large enough to store the entire data source, preferring higher read speed
  - 10 Gigabit network card (capable of transferring at 300 MB/s)
  - `tidb-lightning` fully consumes all CPU cores when running, and deploying on a dedicated machine is highly recommended. If not possible, `tidb-lightning` could be deployed together with other components like `tidb-server`, and the CPU usage could be limited via the `region-concurrency` setting.
- **`tikv-importer`:**
  - 32+ logical cores CPU
  - 40 GB+ memory
  - 1 TB+ SSD, preferring higher IOPS ( 8000 is recommended)
    - \* The disk should be larger than the total size of the top N tables, where  $N = \max(\text{index-concurrency}, \text{table-concurrency})$ .
  - 10 Gigabit network card (capable of transferring at 300 MB/s)
  - `tikv-importer` fully consumes all CPU, disk I/O and network bandwidth when running, and deploying on a dedicated machine is strongly recommended.

If you have sufficient machines, you can deploy multiple `tidb lightning + tikv → importer` servers, with each working on a distinct set of tables, to import the data in parallel.

Deploy TiDB Lightning manually

Step 1: Deploy a TiDB cluster

Before importing data, you need to have a deployed TiDB cluster, with the cluster version 2.0.9 or above. It is highly recommended to use the latest version.

You can find deployment instructions in [TiDB Quick Start Guide](#).

Step 2: Download the TiDB Lightning installation package

Refer to the [TiDB enterprise tools download page](#) to download the TiDB Lightning package (choose the same version as that of the TiDB cluster).

Step 3: Start `tikv-importer`

1. Upload `bin/tikv-importer` from the installation package.
2. Configure `tikv-importer.toml`.

```
# TiKV Importer configuration file template

# Log file
log-file = "tikv-importer.log"
# Log level: trace, debug, info, warn, error, off.
log-level = "info"

# Listening address of the status server.
status-server-address = "0.0.0.0:8286"

[server]
# The listening address of tikv-importer. tidb-lightning needs to
# → connect to
# this address to write data.
addr = "0.0.0.0:8287"

[import]
# The directory to store engine files.
import-dir = "/mnt/ssd/data.import/"
```

The above only shows the essential settings. See the [Configuration](#) section for the full list of settings.

3. Run `tikv-importer`.

```
nohup ./tikv-importer -C tikv-importer.toml > nohup.out &
```

Step 4: Start `tidb-lightning`

1. Upload `bin/tidb-lightning` and `bin/tidb-lightning-ctl` from the tool set.
2. Mount the data source onto the same machine.

3. Configure `tidb-lightning.toml`. For configurations that do not appear in the template below, TiDB Lightning writes a configuration error to the log file and exits.

```
[lightning]
# The concurrency number of data. It is set to the number of logical
# cores by default. When deploying together with other components, you
# set it to 75% of the size of logical CPU cores to limit the CPU usage
# .
# region-concurrency =

# Logging
level = "info"
file = "tidb-lightning.log"

[tikv-importer]
# The listening address of tikv-importer. Change it to the actual
# address.
addr = "172.16.31.10:8287"

[mydumper]
# mydumper local source data directory
data-source-dir = "/data/my_database"

[tidb]
# Configuration of any TiDB server from the cluster
host = "172.16.31.1"
port = 4000
user = "root"
password = ""
# Table schema information is fetched from TiDB via this status-port.
status-port = 10080
```

The above only shows the essential settings. See the [Configuration](#) section for the full list of settings.

4. Run `tidb-lightning`. If you directly run the command in the command-line, the process might exit because of the SIGHUP signal received. Instead, it's preferable to run a bash script that contains the `nohup` command:

```
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

#### 10.8.6.5 Use TiDB Lightning to Import Data in Parallel

Since v5.3.0, the [Local-backend mode](#) of TiDB Lightning supports the parallel import of a single table or multiple tables. By simultaneously running multiple TiDB Lightning instances, you can import data in parallel from different single tables or multiple tables. In this way, TiDB Lightning provides the ability to scale horizontally, which can greatly reduce the time required to import large amounts of data.

In technical implementation, TiDB Lightning records the meta data of each instance and the data of each imported table in the target TiDB, and coordinates the Row ID allocation range of different instances, the record of global Checksum, and the configuration changes and recovery of TiKV and PD.

You can use TiDB Lightning to import data in parallel in the following scenarios:

- Import sharded schemas and sharded tables. In this scenario, multiple tables from multiple upstream database instances are imported into the downstream TiDB database by different TiDB Lightning instances in parallel.
- Import single tables in parallel. In this scenario, single tables stored in a certain directory or cloud storage (such as Amazon S3) are imported into the downstream TiDB cluster by different TiDB Lightning instances in parallel. This is a new feature introduced in TiDB 5.3.0.

**Note:**

Parallel import only supports the initialized empty tables in TiDB. It does not support migrating data to tables with data written by existing services. Otherwise, data inconsistencies may occur.

The following diagram shows how importing sharded schemas and sharded tables works. In this scenario, you can use multiple TiDB Lightning instances to import MySQL sharded tables to a downstream TiDB cluster.

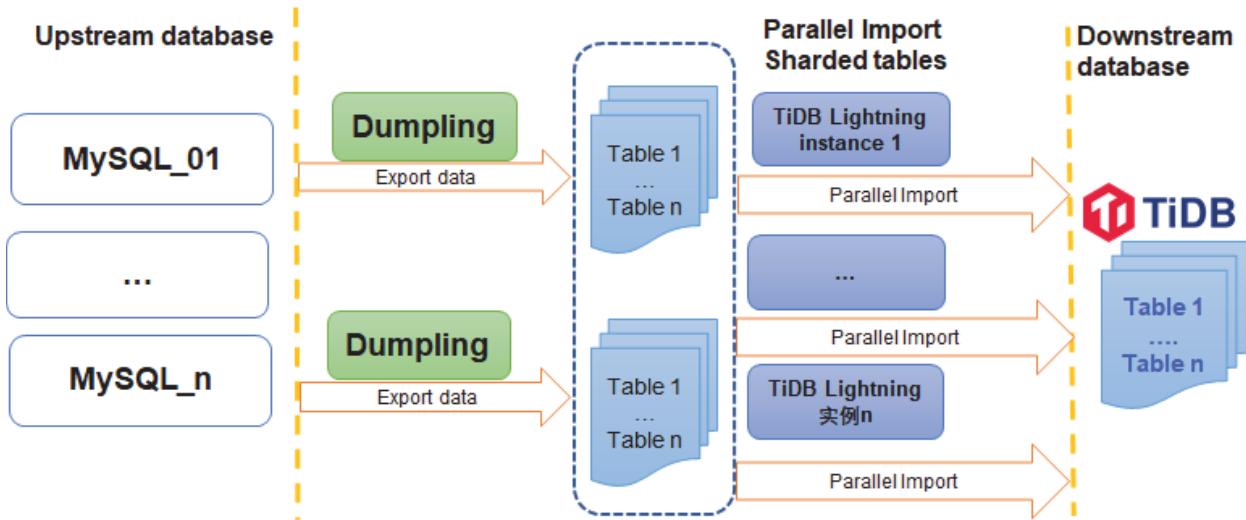


Figure 148: Import sharded schemas and sharded tables

The following diagram shows how importing single tables works. In this scenario, you can use multiple TiDB Lightning instances to split data from a single table and import it in parallel to a downstream TiDB cluster.

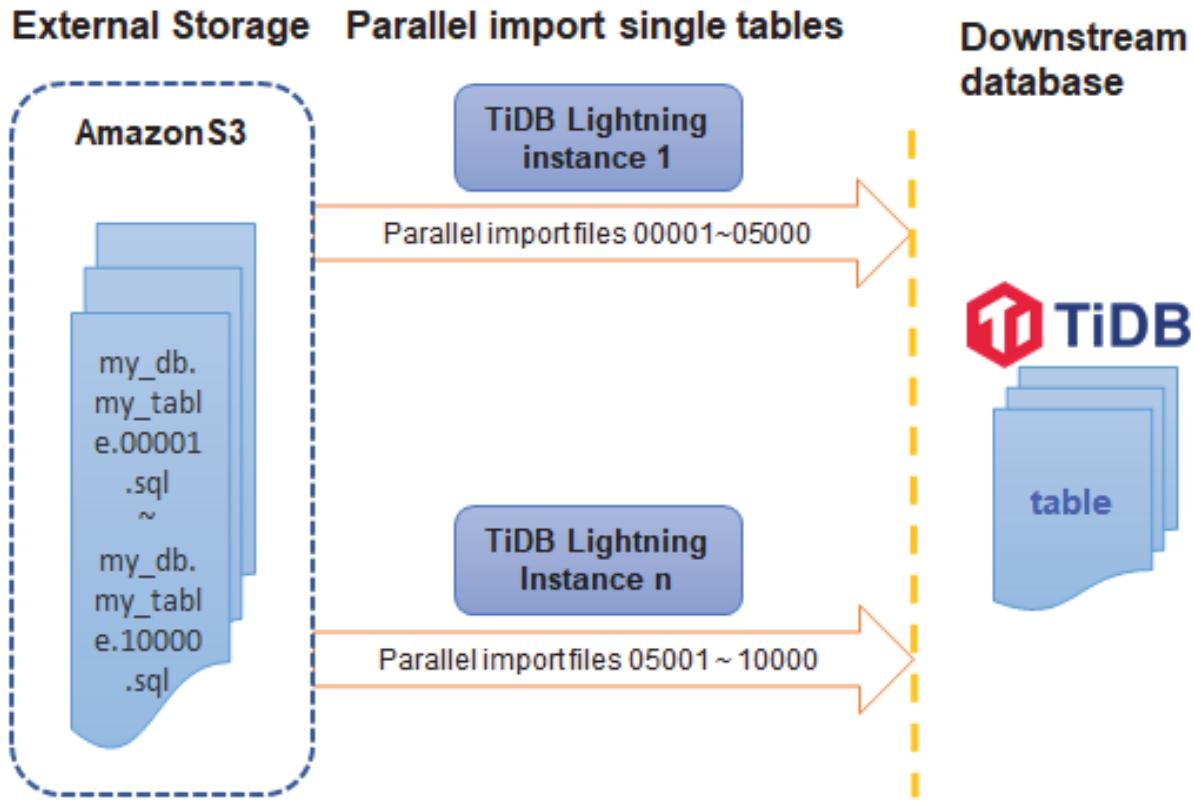


Figure 149: Import single tables

#### 10.8.6.5.1 Considerations

No additional configuration is required for parallel import using TiDB Lightning. When TiDB Lightning is started, it registers meta data in the downstream TiDB cluster and automatically detects whether there are other instances migrating data to the target cluster at the same time. If there is, it automatically enters the parallel import mode.

But when migrating data in parallel, you need to take the following into consideration:

- Handle conflicts between primary keys or unique indexes across multiple sharded tables
- Optimize import performance

Handle conflicts between primary keys or unique indexes

When using **Local-backend mode** to import in parallel, you need to ensure that there are no primary key or unique index conflicts between data sources, and between the tables in the target TiDB cluster. Ensure that there are no data writes in the target table during import. Otherwise, TiDB Lightning will not be able to guarantee the correctness of the imported data, and the target table will contain inconsistent indexes after the import is completed.

Optimize import performance

Because TiDB Lightning needs to upload the generated Key-Value data to the TiKV node where each copy of the corresponding Region is located, the import speed is limited by the size of the target cluster. It is recommended to ensure that the number of TiKV instances in the target TiDB cluster and the number of TiDB Lightning instances are greater than n:1 (n is the number of copies of the Region). At the same time, you need to meet the following requirements to achieve the optimal import performance:

- The total size of source files for each TiDB Lightning instances performing parallel import should be smaller than 5 TiB
- The total number of TiDB Lightning instances should be smaller than 10

When using TiDB Lightning to import shared databases and tables in parallel, choose an appropriate number of TiDB Lightning instances according to the amount of data.

- If the MySQL data volume is less than 2 TiB, you can use one TiDB Lightning instance for parallel import.
- If the MySQL data volume exceeds 2 TiB and the total number of MySQL instance is smaller than 10, it is recommended that you use one TiDB Lightning instance for each MySQL instance, and the number of parallel TiDB Lightning instances should not exceed 10.
- If the MySQL data volume exceeds 2 TiB and the total number of MySQL instance exceeds 10, it is recommended that you allocate 5 to 10 TiDB Lightning instances for importing the data exported by these MySQL instances.

Next, this document uses two examples to detail the operation steps of parallel import in different scenarios:

- Example 1: Use Dumpling + TiDB Lightning to import sharded databases and tables into TiDB in parallel
- Example 2: Import single tables in parallel

#### 10.8.6.5.2 Example 1: Use Dumpling + TiDB Lightning to Import Sharded Databases and Tables into TiDB in Parallel

In this example, assume that the upstream is a MySQL cluster with 10 sharded tables, with a total size of 10 TiB. You can use 5 TiDB Lightning instances to perform parallel import, and each instance imports 2 TiB. It is estimated that the total import time (excluding the time required for Dumpling export) can be reduced from about 40 hours to about 10 hours.

Assume that the upstream library is named `my_db`, and the name of each sharded table is `my_table_01 ~ my_table_10`. You want to merge and import them into the downstream `my_db.my_table` table. The specific steps are described in the following sections.

Step 1: Use Dumpling to export data

Export two sharded tables on the 5 nodes where TiDB Lightning is deployed:

- If the two sharded tables are in the same MySQL instance, you can use the `-f` parameter of Dumpling to directly export them. When using TiDB Lightning to import, you can specify `data-source-dir` as the directory where Dumpling exports data to;
- If the data of the two sharded tables are distributed on different MySQL nodes, you need to use Dumpling to separately export them. The exported data needs to be placed in the same parent directory but in different sub-directories. When using TiDB Lightning to perform parallel import, you need to specify `data-source-dir` as the parent directory.

For more information on how to use Dumpling to export data, see [Dumpling](#).

Step 2: Configure TiDB Lightning data sources

Create a configuration file `tidb-lightning.toml`, and then add the following content:

```
[lightning]
status-addr = ":8289"

[mydumper]
##### Specify the path for Dumpling to export data. If Dumpling performs
    ↪ several times and the data belongs to different directories, you can
    ↪ place all the exported data in the same parent directory and specify
    ↪ this parent directory here.
data-source-dir = "/path/to/source-dir"

[tikv-importer]
##### Use the Local backend mode.
backend = "local"

##### Specify the path for local sorting data.
sorted-kv-dir = "/path/to/sorted-dir"

##### Specify the routes for shard schemas and tables.
[[routes]]
schema-pattern = "my_db"
table-pattern = "my_table_"
target-schema = "my_db"
target-table = "my_table"
```

If the data source is stored in a distributed storage cache such as Amazon S3 or GCS, see [External Storages](#).

Step 3: Start TiDB Lightning to import data

During parallel import, the server configuration requirements for each TiDB Lightning node are the same as the non-parallel import mode. Each TiDB Lightning node needs to consume the same resources. It is recommended to deploy them on different servers. For detailed deployment steps, see [Deploy TiDB Lightning](#).

Start TiDB Lightning on each server in turn. If you use `nohup` to directly start it from the command line, it might exit due to the SIGHUP signal. So it is recommended to put `nohup` in the script, for example:

```
####!/bin/bash  
nohup ./tidb-lightning -config tidb-lightning.toml > nohup.out &
```

During parallel import, TiDB Lightning automatically performs the following checks after starting the task.

- Check whether there is enough space on the local disk and on the TiKV cluster for importing data. TiDB Lightning samples the data sources and estimates the percentage of the index size from the sample result. Because indexes are included in the estimation, there may be cases where the size of the source data is less than the available space on the local disk, but still the check fails.
- Check whether the regions in the TiKV cluster are distributed evenly and whether there are too many empty regions. If the number of empty regions exceeds  $\max(1000, \text{number of tables} * 3)$ , i.e. greater than the bigger one of “1000” or “3 times the number of tables”, then the import cannot be executed.
- Check whether the data is imported in order from the data sources. The size of `mydumper.batch-size` is automatically adjusted based on the result of the check. Therefore, the `mydumper.batch-size` configuration is no longer available.

You can also turn off the check and perform a forced import with the `lightning.check → -requirements` configuration. For more detailed checks, see [TiDB Lightning prechecks](#)

#### Step 4: Check the import progress

After starting the import, you can check the progress in either of the following ways:

- Check the progress through the `grep` log keyword `progress`. It is updated every 5 minutes by default.
- Check the progress through the monitoring console. For details, see [TiDB Lightning Monitoring](#).

Wait for all TiDB Lightning instances to finish, then the entire import is completed.

#### 10.8.6.5.3 Example 2: Import single tables in parallel

TiDB Lightning also supports parallel import of single tables. For example, import multiple single tables stored in Amazon S3 by different TiDB Lightning instances into the downstream TiDB cluster in parallel. This method can speed up the overall import speed. For more information on external storages, see [External Storages](#)).

### Note:

In the local environment, you can use the `--where` parameter of Dumpling to divide the data of a single table into different parts and export it to the local disks of multiple servers in advance. This way, you can still perform parallel import. The configuration is the same as Example 1.

Assuming that the source files are stored in Amazon S3, the table files are `my_db.`  
 $\hookrightarrow$  `my_table.00001.sql ~ my_db.my_table.10000.sql`, a total of 10,000 SQL files. If you want to use 2 TiDB Lightning instances to speed up the import, you need to add the following settings in the configuration file:

```
[[mydumper.files]]
##### the db schema file
pattern = '(?i)^(?:[^/]*/)*my_db-schema-create\.sql'
schema = "my_db"
type = "schema-schema"

[[mydumper.files]]
##### the table schema file
pattern = '(?i)^(?:[^/]*/)*my_db\.my_table-schema\.sql'
schema = "my_db"
table = "my_table"
type = "table-schema"

[[mydumper.files]]
##### Only import 00001~05000 and ignore other files
pattern = '(?i)^(?:[^/]*/)*my_db\.my_table\.(0[0-4] [0-9] [0-9] [0-9] |05000)\.
    \hookrightarrow sql'
schema = "my_db"
table = "my_table"
type = "sql"
```

You can modify the configuration of the other instance to only import the `05001 ~ 10000` data files.

For other steps, see the relevant steps in Example 1.

#### 10.8.6.6 TiDB Lightning Web Interface

TiDB Lightning provides a webpage for viewing the import progress and performing some simple task management. This is called the *server mode*.

To enable server mode, either start `tidb-lightning` with the `--server-mode` flag

```
./tidb-lightning --server-mode --status-addr :8289
```

or set the `lightning.server-mode` setting in the configuration file.

```
[lightning]
server-mode = true
status-addr = ':8289'
```

After TiDB Lightning is launched, visit `http://127.0.0.1:8289` to control the program (the actual URL depends on the `status-addr` setting).

In server mode, TiDB Lightning does not start running immediately. Rather, users submit (multiple) *tasks* via the web interface to import data.

#### 10.8.6.6.1 Front page

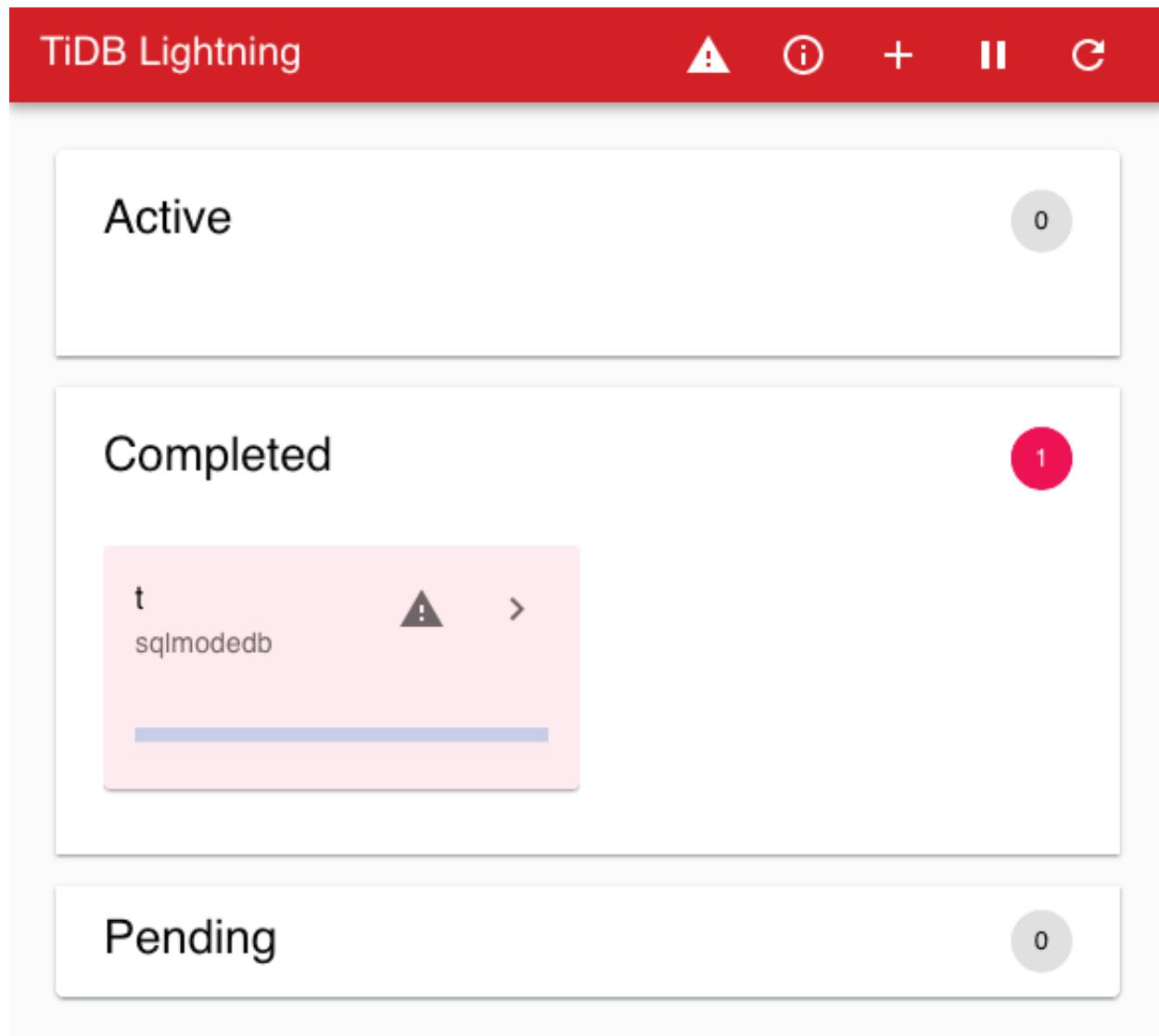


Figure 150: Front page of the web interface

Functions of the title bar, from left to right:

Icon	Function
"TiDB	Click
Light-	to go
ning"	back
	to the
	front
	page

Icon	Function
	Display
	any
	error
	mes-
	sage
	from
	<i>previ-</i>
	<i>ous</i>
	task
	List
	cur-
	rent
	and
	queued
	tasks;
a	
	badge
	may
	ap-
	pear
	here
	to in-
	dicate
	num-
	ber of
	queued
	tasks
+	Submit
	a task
/	Pause/resume
	cur-
	rent
	execu-
	tion
	Configure
	auto-
	refresh
	of the
	web
	page

Three panels below the title bar show all tables in different states:

- Active: these tables are currently being imported
- Completed: these tables have been imported successfully or failed
- Pending: these tables are not yet processed

Each panel contains cards describing the status of the table.

#### 10.8.6.6.2 Submit task

Click the + button on the title bar to submit a task.

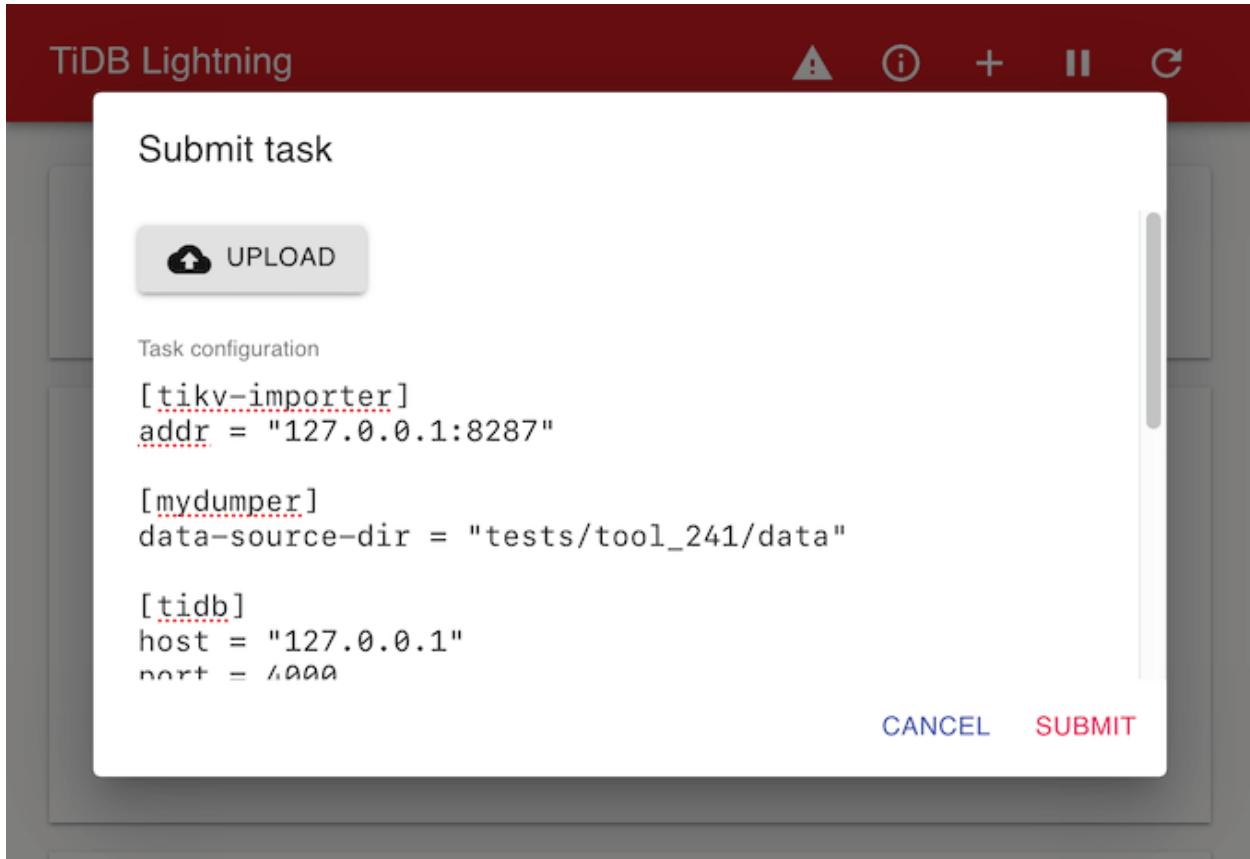


Figure 151: Submit task dialog

Tasks are TOML files described as [task configurations](#). One could also open a local TOML file by clicking **UPLOAD**.

Click **SUBMIT** to run the task. If a task is already running, the new task will be queued and executed after the current task succeeds.

#### 10.8.6.6.3 Table progress

Click the > button of a table card on the front page to view the detailed progress of a table.



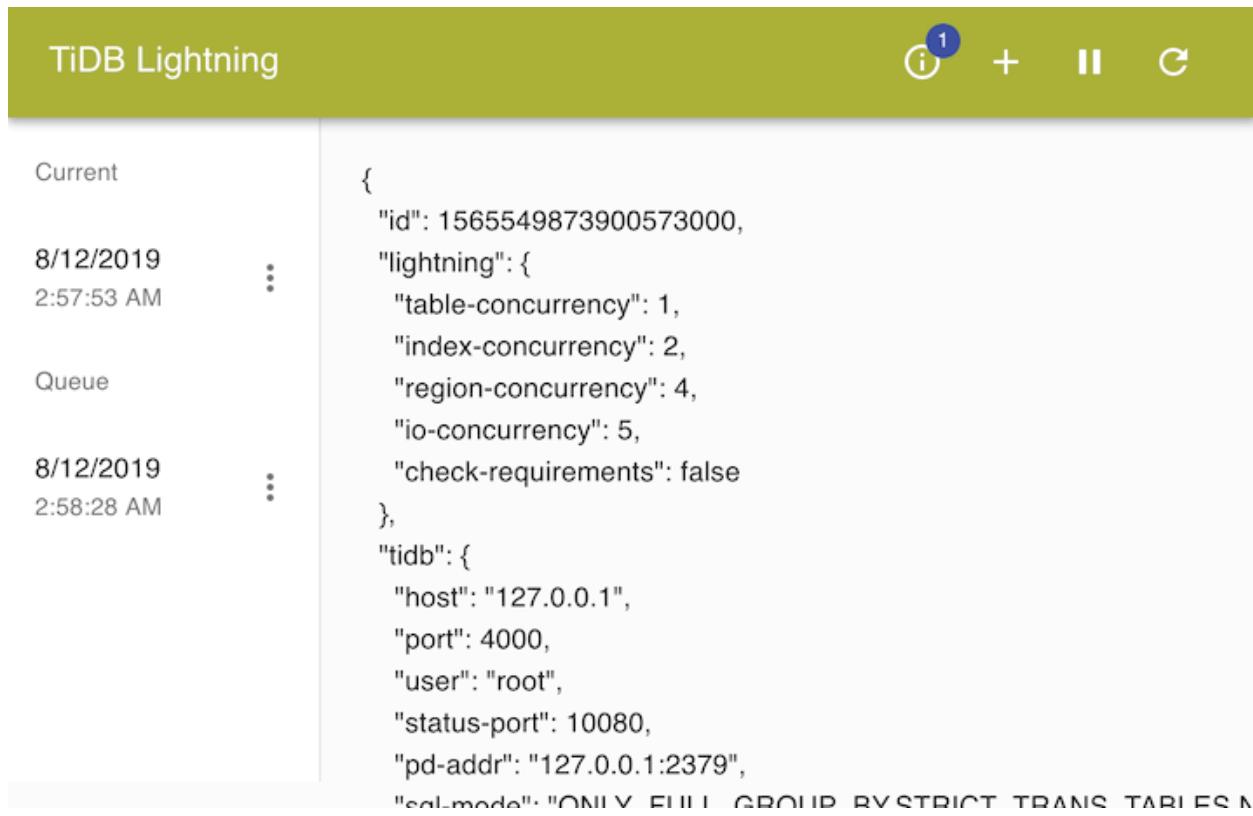
Figure 152: Table progress

The page shows the import progress of every engine and data files associated with the table.

Click **TiDB Lightning** on the title bar to go back to the front page.

#### 10.8.6.6.4 Task management

Click the  button on the title bar to manage the current and queued tasks.



```

{
  "id": 1565549873900573000,
  "lightning": {
    "table-concurrency": 1,
    "index-concurrency": 2,
    "region-concurrency": 4,
    "io-concurrency": 5,
    "check-requirements": false
  },
  "tidb": {
    "host": "127.0.0.1",
    "port": 4000,
    "user": "root",
    "status-port": 10080,
    "pd-addr": "127.0.0.1:2379",
    "sql-mode": "ONLY_FULL_GROUP_BY STRICT_TRANS_TABLES_N"
  }
}

```

Figure 153: Task management page

Each task is labeled by the time it was submitted. Clicking the task would show the configuration formatted as JSON.

Manage tasks by clicking the button next to a task. You can stop a task immediately, or reorder queued tasks.

### 10.8.7 TiDB Lightning Monitoring

`tidb-lightning` supports metrics collection via [Prometheus](#). This document introduces the monitor configuration and monitoring metrics of TiDB Lightning.

#### 10.8.7.1 Monitor configuration

If TiDB Lightning is manually installed, follow the instructions below.

The metrics of `tidb-lightning` can be gathered directly by Prometheus as long as it is discovered. You can set the metrics port in `tidb-lightning.toml`:

```
[lightning]
### HTTP port for debugging and Prometheus metrics pulling (0 to disable)
pprof-port = 8289
```

...

and in `tikv-importer.toml`:

```
### Listening address of the status server.
status-server-address = '0.0.0.0:8286'
```

You need to configure Prometheus to make it discover the servers. For instance, you can directly add the server address to the `scrape_configs` section:

```
...
scrape_configs:
  - job_name: 'tidb-lightning'
    static_configs:
      - targets: ['192.168.20.10:8289']
  - job_name: 'tikv-importer'
    static_configs:
      - targets: ['192.168.20.9:8286']
```

### 10.8.7.2 Grafana dashboard

Grafana is a web interface to visualize Prometheus metrics as dashboards.

When you [deploy a TiDB cluster using TiUP](#) and have added Grafana and Prometheus in the topology configuration, a set of [Grafana + Prometheus monitoring platform](#) is deployed simultaneously. In this situation, you must first import [the JSON file of the dashboard](#).

#### 10.8.7.2.1 Row 1: Speed

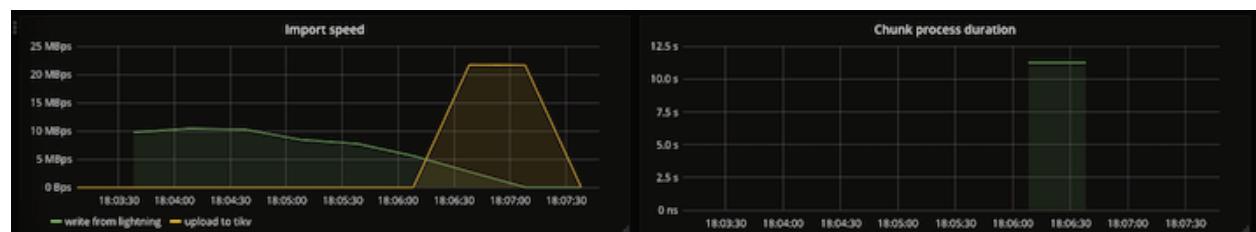


Figure 154: Panels in first row

Panel	Series	Description
Import speed	write from TiDB	Speed of sending KVs from TiDB
Import speed	upload to tikv	Total upload speed from TiKV
Chunk process duration		Average time needed to completely encode one single data file

Sometimes the import speed will drop to zero allowing other parts to catch up. This is

normal.

#### 10.8.7.2.2 Row 2: Progress

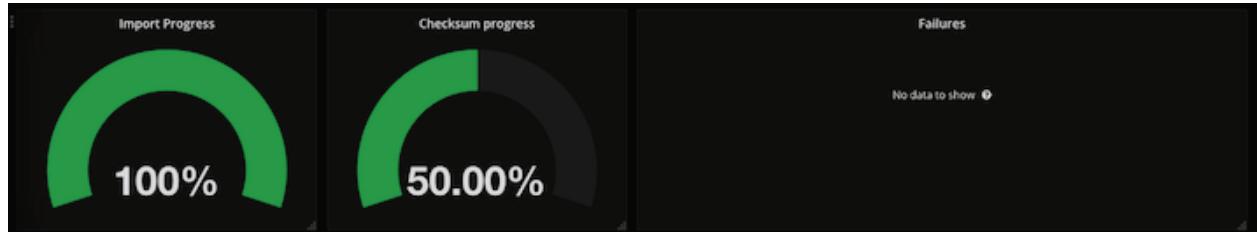


Figure 155: Panels in second row

Panel	Description
Import progress	Percentage of data files encoded so far
Checksum progress	Percentage of tables are verified to be imported successfully
Failures	Number of failed tables and their point of failure, normally empty

#### 10.8.7.2.3 Row 3: Resource

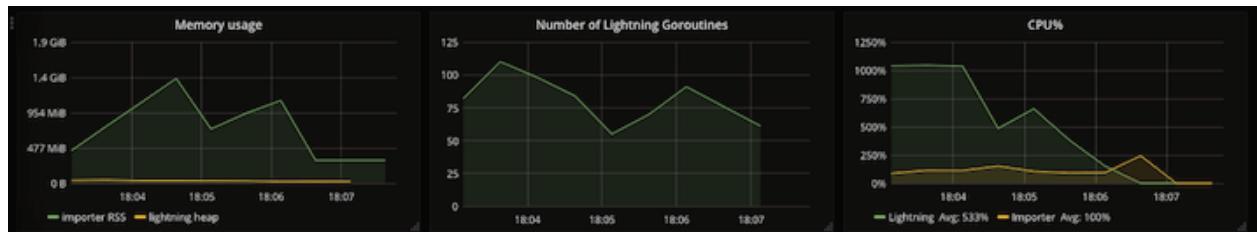


Figure 156: Panels in third row

Panel	Description
Memory usage	Amount of memory occupied by each service

Panel	Description
Number of TiDB Lightning Goroutines	Number of running goroutines used by TiDB Lightning
CPU%	Number of logical CPU cores utilized by each service

#### 10.8.7.2.4 Row 4: Quota



Figure 157: Panels in fourth row

Panel	Series	Description
Idle work- ers	io	Number of unused io- $\hookrightarrow$ concurrency $\hookrightarrow$ , nor- mally close to con- figured value (de- fault 5), and close to 0 means the disk is too slow

Panel	Series	Description
Idle work- ers	closed- engine	Number of engines which is closed but not yet cleaned up, nor- mally close to index + table- concurrency (de- fault 8), and close to 0 means TiDB Light- ning is faster than TiKV Im- porter, which will cause TiDB Light- ning to stall

Panel	Series	Description
Idle workers	table	Number of unused tables $\hookrightarrow$ concurrency $\hookrightarrow$ , normally 0 until the end of process
Idle workers	index	Number of unused indexes $\hookrightarrow$ concurrency $\hookrightarrow$ , normally 0 until the end of process
Idle workers	region	Number of unused regions $\hookrightarrow$ - $\hookrightarrow$ concurrency $\hookrightarrow$ , normally 0 until the end of process

Panel	Series	Description
External re- sources	KV En- coder	Counts active KV en- coders, nor- mally the same as <b>region</b> ↪ - ↪ <b>concurrency</b> ↪ until the end of process
External re- sources	Importer En- gines	Counts opened engine files, should never exceed the <b>max-</b> ↪ <b>open</b> ↪ - ↪ <b>engines</b> ↪ setting

#### 10.8.7.2.5 Row 5: Read speed

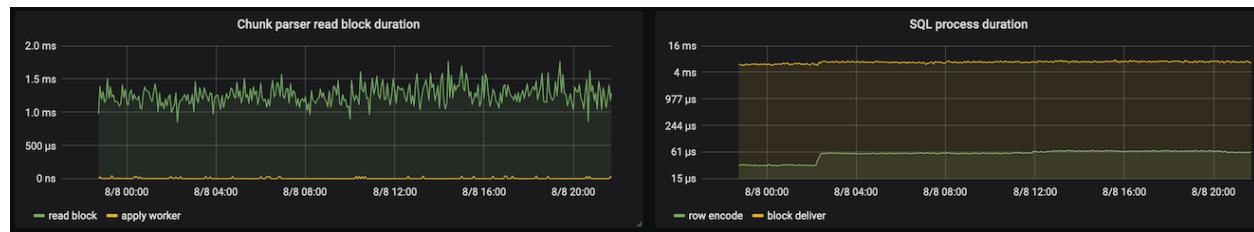


Figure 158: Panels in fifth row

Panel	Series	Description
Chunk parser	read block	Time taken to read one block of bytes to prepare for parsing
Chunk parser	apply worker	Time elapsed to wait for an idle io-concurrency
SQL process	row encode	Time taken to parse and encode a single row
SQL process	block deliver	Time taken to send a block of KV pairs to TiKV Importer

If any of the duration is too high, it indicates that the disk used by TiDB Lightning is too slow or busy with I/O.

#### 10.8.7.2.6 Row 6: Storage



Figure 159: Panels in sixth row

Panel	Series	Description
SQL process rate	data deliver rate	Speed of delivery of data KV pairs to TiKV Im-porter
SQL process rate	index deliver rate	Speed of delivery of index KV pairs to TiKV Im-porter
SQL process rate	total deliver rate	The sum of two rates above

Panel	Series	Description
Total bytes	parser read size	Number of bytes being read by TiDB Lightning
Total bytes	data deliver size	Number of bytes of data KV pairs already delivered to TiKV Importer
Total bytes	index deliver size	Number of bytes of index KV pairs already delivered to TiKV Importer

Panel	Series	Description
Total bytes	storage_size / 3	size occupied by the TiKV cluster, divided by 3 (the default number of replicas)

#### 10.8.7.2.7 Row 7: Import speed



Figure 160: Panels in seventh row

Panel	Series	Description
Delivery duration	Range delivery	Time taken to upload a range of KV pairs to the TiKV cluster
Delivery duration	SST delivery	
SST process duration	Split SST	
SST process duration	SST upload	
SST process duration	SST ingest	
	SST size	Size of SST files in MB, ranging from 0 B to 300 MB

Panel	Series	Description
Delivery duration	SST delivery	Time taken to upload an SST file to the TiKV cluster
SST process duration	Split SST	Time taken to split the stream of KV pairs into SST files
SST process duration	upload	Time taken to upload an SST file
SST process duration	ingest	Time taken to ingest an uploaded SST file
SST process duration	SST size	File size of an SST file

#### 10.8.7.3 Monitoring metrics

This section explains the monitoring metrics of `tikv-importer` and `tidb-lightning`, if you need to monitor other metrics not covered by the default Grafana dashboard.

#### 10.8.7.3.1 tikv-importer

Metrics provided by `tikv-importer` are listed under the namespace `tikv_import_*`.

- **`tikv_import_rpc_duration`** (Histogram)

Bucketed histogram for the duration of an RPC action. Labels:

- `request`: what kind of RPC is executed
  - \* `switch_mode` — switched a TiKV node to import/normal mode
  - \* `open_engine` — opened an engine file
  - \* `write_engine` — received data and written into an engine
  - \* `close_engine` — closed an engine file
  - \* `import_engine` — imported an engine file into the TiKV cluster
  - \* `cleanup_engine` — deleted an engine file
  - \* `compact_cluster` — explicitly compacted the TiKV cluster
  - \* `upload` — uploaded an SST file
  - \* `ingest` — ingested an SST file
  - \* `compact` — explicitly compacted a TiKV node
- `result`: the execution result of the RPC
  - \* `ok`
  - \* `error`

- **`tikv_import_write_chunk_bytes`** (Histogram)

Bucketed histogram for the uncompressed size of a block of KV pairs received from TiDB Lightning.

- **`tikv_import_write_chunk_duration`** (Histogram)

Bucketed histogram for the time needed to receive a block of KV pairs from TiDB Lightning.

- **`tikv_import_upload_chunk_bytes`** (Histogram)

Bucketed histogram for the compressed size of a chunk of SST file uploaded to TiKV.

- **`tikv_import_upload_chunk_duration`** (Histogram)

Bucketed histogram for the time needed to upload a chunk of SST file to TiKV.

- **`tikv_import_range_delivery_duration`** (Histogram)

Bucketed histogram for the time needed to deliver a range of KV pairs into a `dispatch ↵ -job`.

- **`tikv_import_split_sst_duration`** (Histogram)

Bucketed histogram for the time needed to split off a range from the engine file into a single SST file.

- **tikv\_import\_sst\_delivery\_duration** (Histogram)

Bucketed histogram for the time needed to deliver an SST file from a `dispatch-job` to an `ImportSSTJob`.

- **tikv\_import\_sst\_recv\_duration** (Histogram)

Bucketed histogram for the time needed to receive an SST file from a `dispatch-job` in an `ImportSSTJob`.

- **tikv\_import\_sst\_upload\_duration** (Histogram)

Bucketed histogram for the time needed to upload an SST file from an `ImportSSTJob` to a TiKV node.

- **tikv\_import\_sst\_chunk\_bytes** (Histogram)

Bucketed histogram for the compressed size of the SST file uploaded to a TiKV node.

- **tikv\_import\_sst\_ingest\_duration** (Histogram)

Bucketed histogram for the time needed to ingest an SST file into TiKV.

- **tikv\_import\_each\_phase** (Gauge)

Indicates the running phase. Possible values are 1, meaning running inside the phase, and 0, meaning outside the phase. Labels:

– `phase`: `prepare/import`

- **tikv\_import\_wait\_store\_available\_count** (Counter)

Counts the number of times a TiKV node is found to have insufficient space when uploading SST files. Labels:

– `store_id`: The TiKV store ID.

#### 10.8.7.3.2 tidb-lightning

Metrics provided by `tidb-lightning` are listed under the namespace `lightning_*`.

- **lightning\_importer\_engine** (Counter)

Counts open and closed engine files. Labels:

– `type`:

- \* `open`
- \* `closed`

- **lightning\_idle\_workers** (Gauge)

Counts idle workers. Labels:

- **name:**
  - \* **table** — the remainder of **table-concurrency**, normally 0 until the end of the process
  - \* **index** — the remainder of **index-concurrency**, normally 0 until the end of the process
  - \* **region** — the remainder of **region-concurrency**, normally 0 until the end of the process
  - \* **io** — the remainder of **io-concurrency**, normally close to configured value (default 5), and close to 0 means the disk is too slow
  - \* **closed-engine** — number of engines which have been closed but not yet cleaned up, normally close to index + table-concurrency (default 8). A value close to 0 means TiDB Lightning is faster than TiKV Importer, which might cause TiDB Lightning to stall
- **lightning\_kv\_encoder** (Counter)
 

Counts open and closed KV encoders. KV encoders are in-memory TiDB instances that convert SQL INSERT statements into KV pairs. The net values need to be bounded in a healthy situation. Labels:

  - **type:**
    - \* **open**
    - \* **closed**
- **lightning\_tables** (Counter)
 

Counts processed tables and their statuses. Labels:

  - **state:** the status of the table, indicating which phase should be completed
    - \* **pending** — not yet processed
    - \* **written** — all data encoded and sent
    - \* **closed** — all corresponding engine files closed
    - \* **imported** — all engine files have been imported into the target cluster
    - \* **altered\_auto\_inc** — AUTO\_INCREMENT ID altered
    - \* **checksum** — checksum performed
    - \* **analyzed** — statistics analysis performed
    - \* **completed** — the table has been fully imported and verified
  - **result:** the result of the current phase
    - \* **success** — the phase completed successfully
    - \* **failure** — the phase failed (did not complete)
- **lightning\_engines** (Counter)
 

Counts number of engine files processed and their status. Labels:

  - **state:** the status of the engine, indicating which phase should be completed

- \* `pending` — not yet processed
- \* `written` — all data encoded and sent
- \* `closed` — engine file closed
- \* `imported` — the engine file has been imported into the target cluster
- \* `completed` — the engine has been fully imported
- `result`: the result of the current phase
  - \* `success` — the phase completed successfully
  - \* `failure` — the phase failed (did not complete)

- **`lightning_chunks`** (Counter)

Counts number of chunks processed and their status. Labels:

- `state`: a chunk's status, indicating which phase the chunk is in
    - \* `estimated` — (not a state) this value gives total number of chunks in current task
    - \* `pending` — loaded but not yet processed
    - \* `running` — data are being encoded and sent
    - \* `finished` — the entire chunk has been processed
    - \* `failed` — errors happened during processing
  - **`lightning_import_seconds`** (Histogram)
- Bucketed histogram for the time needed to import a table.
- **`lightning_row_read_bytes`** (Histogram)
- Bucketed histogram for the size of a single SQL row.
- **`lightning_row_encode_seconds`** (Histogram)
- Bucketed histogram for the time needed to encode a single SQL row into KV pairs.
- **`lightning_row_kv_deliver_seconds`** (Histogram)
- Bucketed histogram for the time needed to deliver a set of KV pairs corresponding to one single SQL row.
- **`lightning_block_deliver_seconds`** (Histogram)
- Bucketed histogram for the time needed to deliver a block of KV pairs to Importer.
- **`lightning_block_deliver_bytes`** (Histogram)
- Bucketed histogram for the uncompressed size of a block of KV pairs delivered to Importer.
- **`lightning_chunk_parser_read_block_seconds`** (Histogram)
- Bucketed histogram for the time needed by the data file parser to read a block.

- **lightning\_checksum\_seconds** (Histogram)

Bucketed histogram for the time needed to compute the checksum of a table.

- **lightning\_apply\_worker\_seconds** (Histogram)

Bucketed histogram for the time needed to acquire an idle worker (see also the `lightning_idle_workers` gauge). Labels:

- **name**:
  - \* `table`
  - \* `index`
  - \* `region`
  - \* `io`
  - \* `closed-engine`

## 10.8.8 TiDB Lightning FAQs

### 10.8.8.1 What is the minimum TiDB/TiKV/PD cluster version supported by TiDB Lightning?

The version of TiDB Lightning should be the same as the cluster. If you use the Local-backend mode, the earliest available version is 4.0.0. If you use the Importer-backend mode or the TiDB-backend mode, the earliest available version is 2.0.9, but it is recommended to use the 3.0 stable version.

### 10.8.8.2 Does TiDB Lightning support importing multiple schemas (databases)?

Yes.

### 10.8.8.3 What is the privilege requirements for the target database?

TiDB Lightning requires the following privileges:

- SELECT
- UPDATE
- ALTER
- CREATE
- DROP

If the `TiDB-backend` is chosen, or the target database is used to store checkpoints, it additionally requires these privileges:

- INSERT
- DELETE

The Local-backend and Importer-backend do not require these two privileges because data is ingested into TiKV directly, which bypasses the entire TiDB privilege system. This is secure as long as the ports of TiKV, TiKV Importer and TiDB Lightning are not reachable outside the cluster.

If the `checksum` configuration of TiDB Lightning is set to `true`, then the admin user privileges in the downstream TiDB need to be granted to TiDB Lightning.

#### 10.8.8.4 TiDB Lightning encountered an error when importing one table. Will it affect other tables? Will the process be terminated?

If only one table has an error encountered, the rest will still be processed normally.

#### 10.8.8.5 How to properly restart TiDB Lightning?

If you are using Importer-backend, depending on the status of `tikv-importer`, the basic sequence of restarting TiDB Lightning is like this:

If `tikv-importer` is still running:

1. Stop `tidb-lightning`.
2. Perform the intended modifications, such as fixing the source data, changing settings, replacing hardware etc.
3. If the modification previously has changed any table, remove the corresponding checkpoint too.
4. Start `tidb-lightning`.

If `tikv-importer` needs to be restarted:

1. Stop `tidb-lightning`.
2. Stop `tikv-importer`.
3. Perform the intended modifications, such as fixing the source data, changing settings, replacing hardware etc.
4. Start `tikv-importer`.
5. Start `tidb-lightning` and wait until the program fails with *CHECKSUM* error, if any.
  - Restarting `tikv-importer` would destroy all engine files still being written, but `tidb-lightning` did not know about it. As of v3.0 the simplest way is to let `tidb-lightning` go on and retry.
6. Destroy the failed tables and checkpoints
7. Start `tidb-lightning` again.

If you are using Local-backend or TiDB-backend, the operations are the same as those of using Importer-backend when the `tikv-importer` is still running.

#### 10.8.8.6 How to ensure the integrity of the imported data?

TiDB Lightning by default performs checksum on the local data source and the imported tables. If there is checksum mismatch, the process would be aborted. These checksum information can be read from the log.

You could also execute the `ADMIN CHECKSUM TABLE` SQL command on the target table to recompute the checksum of the imported data.

```
ADMIN CHECKSUM TABLE `schema`.`table`;
```

Db_name	Table_name	Checksum_crc64_xor	Total_kvs	Total_bytes
schema	table	5505282386844578743	3	96

1 row in set (0.01 sec)

#### 10.8.8.7 What kind of data source format is supported by TiDB Lightning?

TiDB Lightning only supports the SQL dump generated by [Dumpling](#) or [CSV](#) files stored in the local file system.

#### 10.8.8.8 Could TiDB Lightning skip creating schema and tables?

Yes. If you have already created the tables in the target database, you could set `no-schema = true` in the `[mydumper]` section in `tidb-lightning.toml`. This makes TiDB Lightning skip the `CREATE TABLE` invocations and fetch the metadata directly from the target database. TiDB Lightning will exit with error if a table is actually missing.

#### 10.8.8.9 Can the Strict SQL Mode be disabled to allow importing invalid data?

Yes. By default, the `sql_mode` used by TiDB Lightning is `"STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION"`, which disallows invalid data such as the date `1970-00-00`. The mode can be changed by modifying the `sql-mode` setting in the `[tidb]` section in `tidb-lightning.toml`.

```
...
[tidb]
sql-mode = ""
...
```

#### 10.8.8.10 Can one `tikv-importer` serve multiple `tidb-lightning` instances?

Yes, as long as every `tidb-lightning` instance operates on different tables.

#### 10.8.8.11 How to stop the `tikv-importer` process?

To stop the `tikv-importer` process, you can choose the corresponding operation according to your deployment method.

- For manual deployment: if `tikv-importer` is running in foreground, press Ctrl+C to exit. Otherwise, obtain the process ID using the `ps aux | grep tikv-importer` command and then terminate the process using the `kill <pid>` command.

#### 10.8.8.12 How to stop the `tidb-lightning` process?

To stop the `tidb-lightning` process, you can choose the corresponding operation according to your deployment method.

- For manual deployment: if `tidb-lightning` is running in foreground, press Ctrl+C to exit. Otherwise, obtain the process ID using the `ps aux | grep tidb-lightning` command and then terminate the process using the `kill -2 <pid>` command.

#### 10.8.8.13 Why the `tidb-lightning` process suddenly quits while running in background?

It is potentially caused by starting `tidb-lightning` incorrectly, which causes the system to send a SIGHUP signal to stop the `tidb-lightning` process. In this situation, `tidb-lightning.log` usually outputs the following log:

```
[2018/08/10 07:29:08.310 +08:00] [INFO] [main.go:41] ["got signal to exit"]
  ↳ [signal=hangup]
```

It is not recommended to directly use `nohup` in the command line to start `tidb-lightning`. You can [start `tidb-lightning`](#) by executing a script.

In addition, if the last log of TiDB Lightning shows that the error is “Context canceled”, you need to search for the first “ERROR” level log. This “ERROR” level log is usually followed by “got signal to exit”, which indicates that TiDB Lightning received an interrupt signal and then exited.

#### 10.8.8.14 Why my TiDB cluster is using lots of CPU resources and running very slowly after using TiDB Lightning?

If `tidb-lightning` abnormally exited, the cluster might be stuck in the “import mode”, which is not suitable for production. The current mode can be retrieved using the following command:

```
tidb-lightning-ctl --fetch-mode
```

You can force the cluster back to “normal mode” using the following command:

```
tidb-lightning-ctl --switch-mode=normal
```

#### 10.8.8.15 Can TiDB Lightning be used with 1-Gigabit network card?

The TiDB Lightning toolset is best used with a 10-Gigabit network card. 1-Gigabit network cards are *not recommended*, especially for `tikv-importer`.

1-Gigabit network cards can only provide a total bandwidth of 120 MB/s, which has to be shared among all target TiKV stores. TiDB Lightning can easily saturate all bandwidth of the 1-Gigabit network and bring down the cluster because PD is unable to be contacted anymore. To avoid this, set an *upload speed limit* in `Importer's configuration`:

```
[import]
### Restricts the total upload speed to TiKV to 100 MB/s or less
upload-speed-limit = "100MB"
```

#### 10.8.8.16 Why TiDB Lightning requires so much free space in the target TiKV cluster?

With the default settings of 3 replicas, the space requirement of the target TiKV cluster is 6 times the size of data source. The extra multiple of “2” is a conservative estimation because the following factors are not reflected in the data source:

- The space occupied by indices
- Space amplification in RocksDB

#### 10.8.8.17 Can TiKV Importer be restarted while TiDB Lightning is running?

No. TiKV Importer stores some information of engines in memory. If `tikv-importer` is restarted, `tidb-lightning` will be stopped due to lost connection. At this point, you need to `destroy the failed checkpoints` as those TiKV Importer-specific information is lost. You can restart TiDB Lightning afterwards.

See also [How to properly restart TiDB Lightning?](#) for the correct sequence.

#### 10.8.8.18 How to completely destroy all intermediate data associated with TiDB Lightning?

1. Delete the checkpoint file.

```
tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-
→ remove=all
```

If, for some reason, you cannot run this command, try manually deleting the file `/tmp/tidb_lightning_checkpoint.pb`.

2. If you are using Local-backend, delete the `sorted-kv-dir` directory in the configuration. If you are using Importer-backend, delete the entire `import` directory on the machine hosting `tikv-importer`.
3. Delete all tables and databases created on the TiDB cluster, if needed.

### 10.8.8.19 Why does TiDB Lightning report the could not find first pair, this shouldn't happen error?

This error occurs possibly because the number of files opened by TiDB Lightning exceeds the system limit when TiDB Lightning reads the sorted local files. In the Linux system, you can use the `ulimit -n` command to confirm whether the value of this system limit is too small. It is recommended that you adjust this value to 1000000 (`ulimit -n 1000000`) during the import.

### 10.8.8.20 Import speed is too slow

Normally it takes TiDB Lightning 2 minutes per thread to import a 256 MB data file. If the speed is much slower than this, there is an error. You can check the time taken for each data file from the log mentioning `restore chunk ... takes`. This can also be observed from metrics on Grafana.

There are several reasons why TiDB Lightning becomes slow:

**Cause 1:** `region-concurrency` is set too high, which causes thread contention and reduces performance.

1. The setting can be found from the start of the log by searching `region-concurrency`.
2. If TiDB Lightning shares the same machine with other services (for example, TiKV Importer), `region-concurrency` must be **manually** set to 75% of the total number of CPU cores.
3. If there is a quota on CPU (for example, limited by Kubernetes settings), TiDB Lightning may not be able to read this out. In this case, `region-concurrency` must also be **manually** reduced.

**Cause 2:** The table schema is too complex.

Every additional index introduces a new KV pair for each row. If there are N indices, the actual size to be imported would be approximately  $(N+1)$  times the size of the Dumpling output. If the indices are negligible, you may first remove them from the schema, and add them back using `CREATE INDEX` after the import is complete.

**Cause 3:** Each file is too large.

TiDB Lightning works the best when the data source is broken down into multiple files of size around 256 MB so that the data can be processed in parallel. If each file is too large, TiDB Lightning might not respond.

If the data source is CSV, and all CSV files have no fields containing newline control characters (U+000A and U+000D), you can turn on “strict format” to let TiDB Lightning automatically split the large files.

```
[mydumper]
strict-format = true
```

**Cause 4:** TiDB Lightning is too old.

Try the latest version! Maybe there is new speed improvement.

#### 10.8.8.21 checksum failed: checksum mismatched remote vs local

**Cause:** The checksum of a table in the local data source and the remote imported database differ. This error has several deeper reasons. You can further locate the reason by checking the log that contains `checksum mismatched`.

The lines that contain `checksum mismatched` provide the information `total_kvs: x ↵ vs y`, where **x** indicates the number of key-value pairs (KV pairs) calculated by the target cluster after the import is completed, and **y** indicates the number of key-value pairs generated by the local data source.

- If **x** is greater, it means that there are more KV pairs in the target cluster.
  - It is possible that this table is not empty before the import and therefore affects the data checksum. It is also possible that TiDB Lightning has previously failed and shut down, but did not restart correctly.
- If **y** is greater, it means that there are more KV pairs in the local data source.
  - If the checksum of the target database is all 0, it means that no import has occurred. It is possible that the cluster is too busy to receive any data.
  - It is possible that the exported data contains duplicate data, such as the UNIQUE and PRIMARY KEYs with duplicate values, or that the downstream table structure is case-insensitive while the data is case-sensitive.
- Other possible reasons
  - If the data source is machine-generated and not backed up by Dumpling, make sure the data conforms to the table limits. For example, the AUTO\_INCREMENT column needs to be positive and not 0.

#### Solutions:

1. Delete the corrupted data using `tidb-lightning-ctl`, check the table structure and the data, and restart TiDB Lightning to import the affected tables again.

```
tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-error
↪ -destroy=all
```

2. Consider using an external database to store the checkpoints (change `[checkpoint] ↪ dsn`) to reduce the target database's load.
3. If TiDB Lightning was improperly restarted, see also the “[How to properly restart TiDB Lightning](#)” section in the FAQ.

#### 10.8.8.22 Checkpoint for ... has invalid status: (error code)

**Cause:** `Checkpoint` is enabled, and TiDB Lightning or TiKV Importer has previously abnormally exited. To prevent accidental data corruption, TiDB Lightning will not start until the error is addressed.

The error code is an integer smaller than 25, with possible values of 0, 3, 6, 9, 12, 14, 15, 17, 18, 20, and 21. The integer indicates the step where the unexpected exit occurs in the import process. The larger the integer is, the later step the exit occurs at.

##### Solutions:

If the error was caused by invalid data source, delete the imported data using `tidb-lightning-ctl` and start Lightning again.

```
tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-error-
    ↪ -destroy=all
```

See the [Checkpoints control](#) section for other options.

#### 10.8.8.23 ResourceTemporarilyUnavailable("Too many open engines ...: ...")

**Cause:** The number of concurrent engine files exceeds the limit specified by `tikv-importer`. This could be caused by misconfiguration. Additionally, if `tidb-lightning` exited abnormally, an engine file might be left at a dangling open state, which could cause this error as well.

##### Solutions:

1. Increase the value of `max-open-engines` setting in `tikv-importer.toml`. This value is typically dictated by the available memory. This could be calculated by using:

Max Memory Usage  $\text{max-open-engines} \times \text{write-buffer-size} \times \text{max-write-buffer-number}$

2. Decrease the value of `table-concurrency + index-concurrency` so it is less than `max-open-engines`.
3. Restart `tikv-importer` to forcefully remove all engine files (default to `./data.import /`). This also removes all partially imported tables, which requires TiDB Lightning to clear the outdated checkpoints.

```
tidb-lightning-ctl --config conf/tidb-lightning.toml --checkpoint-error-
    ↪ -destroy=all
```

#### 10.8.8.24 cannot guess encoding for input file, please convert to UTF-8 manually

**Cause:** TiDB Lightning only recognizes the UTF-8 and GB-18030 encodings for the table schemas. This error is emitted if the file isn't in any of these encodings. It is also

possible that the file has mixed encoding, such as containing a string in UTF-8 and another string in GB-18030, due to historical `ALTER TABLE` executions.

**Solutions:**

1. Fix the schema so that the file is entirely in either UTF-8 or GB-18030.
2. Manually `CREATE` the affected tables in the target database, and then set `[mydumper]` ↪ `no-schema = true` to skip automatic table creation.
3. Set `[mydumper] character-set = "binary"` to skip the check. Note that this might introduce mojibake into the target database.

#### 10.8.8.25 [sql2kv] sql encode error = [types:1292]invalid time format: '{1970 1 1 ...}'

**Cause:** A table contains a column with the `timestamp` type, but the time value itself does not exist. This is either because of DST changes or the time value has exceeded the supported range (Jan 1, 1970 to Jan 19, 2038).

**Solutions:**

1. Ensure TiDB Lightning and the source database are using the same time zone.

When executing TiDB Lightning directly, the time zone can be forced using the `$TZ` environment variable.

```
# Manual deployment, and force Asia/Shanghai.  
TZ='Asia/Shanghai' bin/tidb-lightning -config tidb-lightning.toml
```

2. When exporting data using Mydumper, make sure to include the `--skip-tz-utc` flag.
3. Ensure the entire cluster is using the same and latest version of `tzdata` (version 2018i or above).

On CentOS, run `yum info tzdata` to check the installed version and whether there is an update. Run `yum upgrade tzdata` to upgrade the package.

#### 10.8.8.26 [Error 8025: entry too large, the max entry size is 6291456]

**Cause:** A single row of key-value pairs generated by TiDB Lightning exceeds the limit set by TiDB.

**Solution:**

Currently, the limitation of TiDB cannot be bypassed. You can only ignore this table to ensure the successful import of other tables.

#### 10.8.8.27 Encounter rpc error: code = Unimplemented ... when TiDB Lightning switches the mode

**Cause:** Some node(s) in the cluster does not support `switch-mode`. For example, if the TiFlash version is earlier than v4.0.0-rc.2, `switch-mode` is not supported.

##### Solutions:

- If there are TiFlash nodes in the cluster, you can update the cluster to v4.0.0-rc.2 or higher versions.
- Temporarily disable TiFlash if you do not want to upgrade the cluster.

#### 10.8.8.28 tidb lightning encountered error: TiDB version too old, expected '>=4.0.0', found '3.0.18'

TiDB Lightning Local-backend only supports importing data to TiDB clusters of v4.0.0 and later versions. If you try to use Local-backend to import data to a v2.x or v3.x cluster, the above error is reported. At this time, you can modify the configuration to use Importer-backend or TiDB-backend for data import.

Some `nightly` versions might be similar to v4.0.0-beta.2. These `nightly` versions of TiDB Lightning actually support Local-backend. If you encounter this error when using a `nightly` version, you can skip the version check by setting the configuration `check->requirements = false`. Before setting this parameter, make sure that the configuration of TiDB Lightning supports the corresponding version; otherwise, the import might fail.

#### 10.8.8.29 restore table test.district failed: unknown columns in header [...]

This error occurs usually because the CSV data file does not contain a header (the first row is not column names but data). Therefore, you need to add the following configuration to the TiDB Lightning configuration file:

```
[mydumper.csv]
header = false
```

#### 10.8.8.30 How to get the runtime goroutine information of TiDB Lightning

1. If `status-port` has been specified in the configuration file of TiDB Lightning, skip this step. Otherwise, you need to send the USR1 signal to TiDB Lightning to enable `status-port`.

Get the process ID (PID) of TiDB Lightning using commands like `ps`, and then run the following command:

```
kill -USR1 <lightning-pid>
```

Check the log of TiDB Lightning. The log of `starting HTTP server / start HTTP ↵ server / started HTTP server` shows the newly enabled `status-port`.

2. Access `http://<lightning-ip>:<status-port>/debug/pprof/goroutine?debug=2` to get the goroutine information.

## 10.8.9 TiDB Lightning Glossary

This page explains the special terms used in TiDB Lightning's logs, monitoring, configurations, and documentation.

### 10.8.9.1 A

#### 10.8.9.1.1 Analyze

An operation to rebuild the `statistics` information of a TiDB table, which is running the `ANALYZE TABLE` statement.

Because TiDB Lightning imports data without going through TiDB, the statistics information is not automatically updated. Therefore, TiDB Lightning explicitly analyzes every table after importing. This step can be omitted by setting the `post-restore.analyze` configuration to `false`.

#### 10.8.9.1.2 AUTO\_INCREMENT\_ID

Every table has an associated `AUTO_INCREMENT_ID` counter to provide the default value of an auto-incrementing column. In TiDB, this counter is additionally used to assign row IDs.

Because TiDB Lightning imports data without going through TiDB, the `AUTO_INCREMENT_ID ↵` counter is not automatically updated. Therefore, TiDB Lightning explicitly alters `AUTO_INCREMENT_ID` to a valid value. This step is always performed, even if the table has no `AUTO_INCREMENT` columns.

### 10.8.9.2 B

#### 10.8.9.2.1 Back end

Back end is the destination where TiDB Lightning sends the parsed result. Also spelled as “backend”.

See [TiDB Lightning Backends](#) for details.

### 10.8.9.3 C

#### 10.8.9.3.1 Checkpoint

TiDB Lightning continuously saves its progress into a local file or a remote database while importing. This allows it to resume from an intermediate state should it crashes in the process. See the [Checkpoints](#) section for details.

#### 10.8.9.3.2 Checksum

In TiDB Lightning, the checksum of a table is a set of 3 numbers calculated from the content of each KV pair in that table. These numbers are respectively:

- the number of KV pairs,
- total length of all KV pairs, and
- the bitwise-XOR of [CRC-64-ECMA](#) value each pair.

TiDB Lightning [validates the imported data](#) by comparing the [local](#) and [remote checksums](#) of every table. The program would stop if any pair does not match. You can skip this check by setting the `post-restore.checksum` configuration to `false`.

See also the [FAQs](#) for how to properly handle checksum mismatch.

#### 10.8.9.3.3 Chunk

A continuous range of source data, normally equivalent to a single file in the data source.

When a file is too large, TiDB Lightning might split a file into multiple chunks.

#### 10.8.9.3.4 Compaction

An operation that merges multiple small SST files into one large SST file, and cleans up the deleted entries. TiKV automatically compacts data in background while TiDB Lightning is importing.

##### Note:

For legacy reasons, you can still configure TiDB Lightning to explicitly trigger a compaction every time a table is imported. However, this is not recommended and the corresponding settings are disabled by default.

See [RocksDB's wiki page on Compaction](#) for its technical details.

#### 10.8.9.4 D

#### 10.8.9.4.1 Data engine

An [engine](#) for sorting actual row data.

When a table is very large, its data is placed into multiple data engines to improve task pipelining and save space of TiKV Importer. By default, a new data engine is opened for every 100 GB of SQL data, which can be configured through the `mydumper.batch-size` setting.

TiDB Lightning processes multiple data engines concurrently. This is controlled by the `lightning.table-concurrency` setting.

### 10.8.9.5 E

#### 10.8.9.5.1 Engine

In TiKV Importer, an engine is a RocksDB instance for sorting KV pairs.

TiDB Lightning transfers data to TiKV Importer through engines. It first opens an engine, sends KV pairs to it (with no particular order), and finally closes the engine. The engine sorts the received KV pairs after it is closed. These closed engines can then be further uploaded to the TiKV stores for ingestion.

Engines use TiKV Importer's `import-dir` as temporary storage, which are sometimes referred to as "engine files".

See also [data engine](#) and [index engine](#).

### 10.8.9.6 F

#### 10.8.9.6.1 Filter

A configuration list that specifies which tables to be imported or excluded.

See [Table Filter](#) for details.

### 10.8.9.7 I

#### 10.8.9.7.1 Import mode

A configuration that optimizes TiKV for writing at the cost of degraded read speed and space usage.

TiDB Lightning automatically switches to and off the import mode while running. However, if TiKV gets stuck in import mode, you can use `tidb-lightning-ctl` to [force revert](#) to [normal mode](#).

#### 10.8.9.7.2 Index engine

An [engine](#) for sorting indices.

Regardless of number of indices, every table is associated with exactly one index engine.

TiDB Lightning processes multiple index engines concurrently. This is controlled by the `lightning.index-concurrency` setting. Since every table has exactly one index engine, this also configures the maximum number of tables to process at the same time.

#### 10.8.9.7.3 Ingest

An operation which inserts the entire content of an [SST file](#) into the RocksDB (TiKV) store.

Ingestion is a very fast operation compared with inserting KV pairs one by one. This operation is the determinant factor for the performance of TiDB Lightning.

See [RocksDB's wiki page on Creating and Ingesting SST files](#) for its technical details.

### 10.8.9.8 K

#### 10.8.9.8.1 KV pair

Abbreviation of “key-value pair”.

#### 10.8.9.8.2 KV encoder

A routine which parses SQL or CSV rows to KV pairs. Multiple KV encoders run in parallel to speed up processing.

### 10.8.9.9 L

#### 10.8.9.9.1 Local checksum

The [checksum](#) of a table calculated by TiDB Lightning itself before sending the KV pairs to TiKV Importer.

### 10.8.9.10 N

#### 10.8.9.10.1 Normal mode

The mode where [import mode](#) is disabled.

### 10.8.9.11 P

### 10.8.9.11.1 Post-processing

The period of time after the entire data source is parsed and sent to TiKV Importer. TiDB Lightning is waiting for TiKV Importer to upload and [ingest](#) the [SST files](#).

## 10.8.9.12 R

### 10.8.9.12.1 Remote checksum

The [checksum](#) of a table calculated by TiDB after it has been imported.

## 10.8.9.13 S

### 10.8.9.13.1 Scattering

An operation that randomly reassigned the leader and the peers of a [Region](#). Scattering ensures that the imported data are distributed evenly among TiKV stores. This reduces stress on PD.

### 10.8.9.13.2 Splitting

An engine is typically very large (around 100 GB), which is not friendly to TiKV if treated as a single [region](#). TiKV Importer splits an engine into multiple small [SST files](#) (configurable by TiKV Importer's `import.region-split-size` setting) before uploading.

### 10.8.9.13.3 SST file

SST is the abbreviation of “sorted string table”. An SST file is RocksDB’s (and thus TiKV’s) native storage format of a collection of KV pairs.

TiKV Importer produces SST files from a closed [engine](#). These SST files are uploaded and then [ingested](#) into TiKV stores.

## 10.9 TiDB Data Migration

[TiDB Data Migration](#) (DM) is an integrated data migration task management platform, which supports the full data migration and the incremental data replication from MySQL-compatible databases (such as MySQL, MariaDB, and Aurora MySQL) into TiDB. DM can help simplify the data migration process and reduce the operation cost of data migration.

### 10.9.1 DM versions

The stable versions of DM include v1.0, v2.0, and v5.3. It is recommended to use DM v5.3 (the latest stable version of DM) and not recommended to use v1.0 (the earliest stable version of DM).

Currently, the DM documentation is independent of the TiDB documentation. To access the DM documentation, click one of the following links:

- [DM v5.3 documentation](#)
- [DM v2.0 documentation](#)
- [DM v1.0 documentation](#)

#### Note:

- Since October 2021, DM's GitHub repository has been moved to [pingcap/tiflow](#). If you see any issues with DM, submit your issue to the [pingcap/tiflow](#) repository for feedback.
- In earlier versions (v1.0 and v2.0), DM uses version numbers that are independent of TiDB. Since v5.3, DM uses the same version number as TiDB. The next version of DM v2.0 is DM v5.3. There are no compatibility changes from DM v2.0 to v5.3, and the upgrade process is no different from a normal upgrade, only an increase in version number.

#### 10.9.2 Basic features

This section describes the basic data migration features provided by DM.

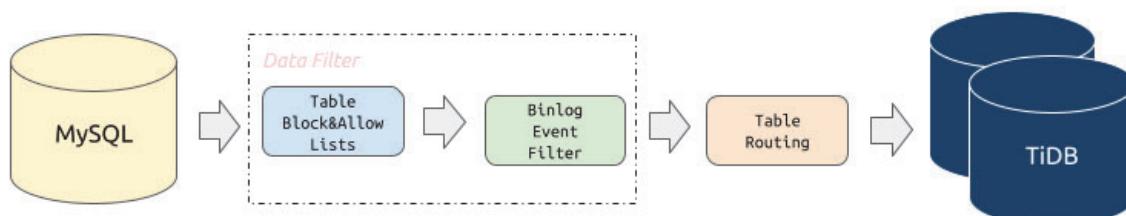


Figure 161: DM Core Features

### 10.9.2.1 Block and allow lists migration at the schema and table levels

The [block and allow lists filtering rule](#) is similar to the `replication-rules-db` → `/replication-rules-table` feature of MySQL, which can be used to filter or replicate all operations of some databases only or some tables only.

### 10.9.2.2 Binlog event filtering

The [binlog event filtering](#) feature means that DM can filter certain types of SQL statements from certain tables in the source database. For example, you can filter all `INSERT` statements in the table `test.sbtest` or filter all `TRUNCATE TABLE` statements in the schema `test`.

### 10.9.2.3 Schema and table routing

The [schema and table routing](#) feature means that DM can migrate a certain table of the source database to the specified table in the downstream. For example, you can migrate the table structure and data from the table `test.sbtest1` in the source database to the table `test.sbtest2` in TiDB. This is also a core feature for merging and migrating sharded databases and tables.

## 10.9.3 Advanced features

### 10.9.3.1 Shard merge and migration

DM supports merging and migrating the original sharded instances and tables from the source databases into TiDB, but with some restrictions. For details, see [Sharding DDL usage restrictions in the pessimistic mode](#) and [Sharding DDL usage restrictions in the optimistic mode](#).

### 10.9.3.2 Optimization for third-party online-schema-change tools in the migration process

In the MySQL ecosystem, tools such as `gh-ost` and `pt-osc` are widely used. DM provides support for these tools to avoid migrating unnecessary intermediate data. For details, see [Online DDL Tools](#)

### 10.9.3.3 Filter certain row changes using SQL expressions

In the phase of incremental replication, DM supports the configuration of SQL expressions to filter out certain row changes, which lets you replicate the data with a greater granularity. For more information, refer to [Filter Certain Row Changes Using SQL Expressions](#).

#### 10.9.4 Usage restrictions

Before using the DM tool, note the following restrictions:

- Database version requirements
  - MySQL version > 5.5
  - MariaDB version >= 10.1.2

**Note:**

If there is a primary-secondary migration structure between the upstream MySQL/MariaDB servers, then choose the following version.

- MySQL version > 5.7.1
- MariaDB version >= 10.1.3

**Warning:**

Migrating data from MySQL 8.0 to TiDB using DM is an experimental feature (introduced since DM v2.0). It is **NOT** recommended that you use it in a production environment.

- DDL syntax compatibility
  - Currently, TiDB is not compatible with all the DDL statements that MySQL supports. Because DM uses the TiDB parser to process DDL statements, it only supports the DDL syntax supported by the TiDB parser. For details, see [MySQL Compatibility](#).
  - DM reports an error when it encounters an incompatible DDL statement. To solve this error, you need to manually handle it using dmctl, either skipping this DDL statement or replacing it with a specified DDL statement(s). For details, see [Skip or replace abnormal SQL statements](#).
- Sharding merge with conflicts
  - If conflict exists between sharded tables, solve the conflict by referring to [handling conflicts of auto-increment primary key](#). Otherwise, data migration is not supported. Conflicting data can cover each other and cause data loss.
  - For other sharding DDL migration restrictions, see [Sharding DDL usage restrictions in the pessimistic mode](#) and [Sharding DDL usage restrictions in the optimistic mode](#).

- Switch of MySQL instances for data sources

When DM-worker connects the upstream MySQL instance via a virtual IP (VIP), if you switch the VIP connection to another MySQL instance, DM might connect to the new and old MySQL instances at the same time in different connections. In this situation, the binlog migrated to DM is not consistent with other upstream status that DM receives, causing unpredictable anomalies and even data damage. To make necessary changes to DM manually, see [Switch DM-worker connection via virtual IP](#).

## 10.10 TiCDC

### 10.10.1 TiCDC Overview

TiCDC is a tool for replicating the incremental data of TiDB. This tool is implemented by pulling TiKV change logs. It can restore data to a consistent state with any upstream TSO, and provides [TiCDC Open Protocol](#) to support other systems to subscribe to data changes.

#### 10.10.1.1 TiCDC Architecture

When TiCDC is running, it is a stateless node that achieves high availability through etcd in PD. The TiCDC cluster supports creating multiple replication tasks to replicate data to multiple different downstream platforms.

The architecture of TiCDC is shown in the following figure:

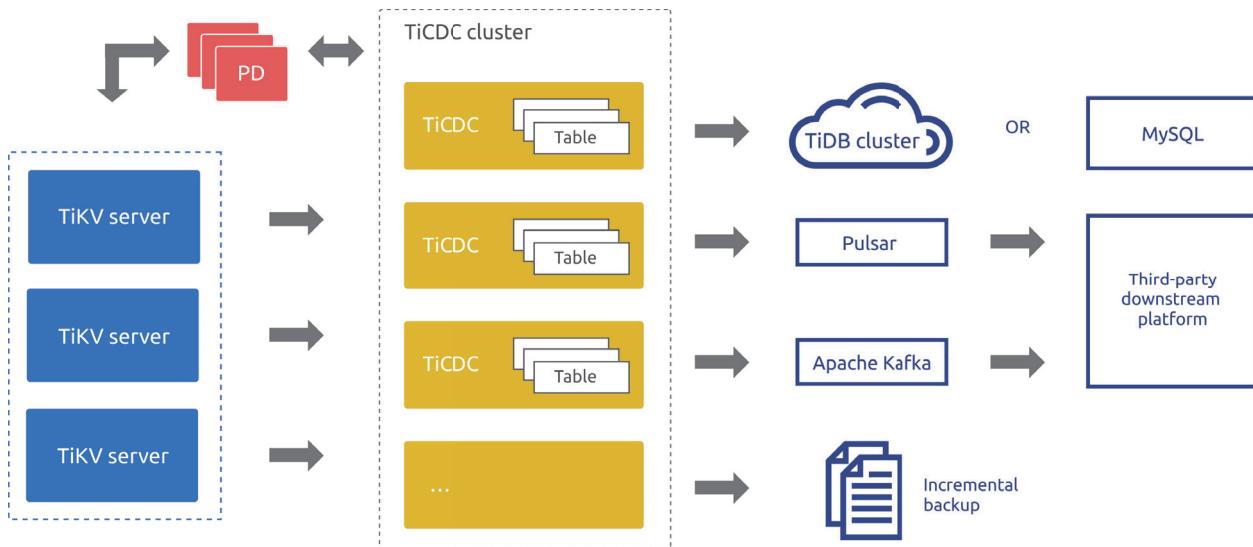


Figure 162: TiCDC architecture

#### 10.10.1.1.1 System roles

- TiKV CDC component: Only outputs key-value (KV) change logs.
  - Assembles KV change logs in the internal logic.
  - Provides the interface to output KV change logs. The data sent includes real-time change logs and incremental scan change logs.
- **capture**: The operating process of TiCDC. Multiple **captures** form a TiCDC cluster that replicates KV change logs.
  - Each **capture** pulls a part of KV change logs.
  - Sorts the pulled KV change log(s).
  - Restores the transaction to downstream or outputs the log based on the TiCDC open protocol.

#### 10.10.1.2 Replication features

This section introduces the replication features of TiCDC.

##### 10.10.1.2.1 Sink support

Currently, the TiCDC sink component supports replicating data to the following downstream platforms:

- Databases compatible with MySQL protocol. The sink component provides the final consistency support.
- Kafka based on the TiCDC Open Protocol. The sink component ensures the row-level order, final consistency or strict transactional consistency.
- `cdclog` (experimental): Files written on the local filesystem or on the Amazon S3-compatible storage.
- Apache Pulsar (experimental)

##### 10.10.1.2.2 Ensure replication order and consistency

Replication order

- For all DDL or DML statements, TiCDC outputs them **at least once**.
- When the TiKV or TiCDC cluster encounters failure, TiCDC might send the same DDL/DML statement repeatedly. For duplicated DDL/DML statements:
  - MySQL sink can execute DDL statements repeatedly. For DDL statements that can be executed repeatedly in the downstream, such as `truncate table`, the statement is executed successfully. For those that cannot be executed repeatedly, such as `create table`, the execution fails, and TiCDC ignores the error and continues the replication.

- Kafka sink sends messages repeatedly, but the duplicate messages do not affect the constraints of `Resolved Ts`. Users can filter the duplicated messages from Kafka consumers.

Replication consistency

- MySQL sink
  - TiCDC does not split single-table transactions and **ensures** the atomicity of single-table transactions.
  - TiCDC does **not ensure** that the execution order of downstream transactions is the same as that of upstream transactions.
  - TiCDC splits cross-table transactions in the unit of table and does **not ensure** the atomicity of cross-table transactions.
  - TiCDC **ensures** that the order of single-row updates is consistent with that in the upstream.
- Kafka sink
  - TiCDC provides different strategies for data distribution. You can distribute data to different Kafka partitions based on the table, primary key, or timestamp.
  - For different distribution strategies, the different consumer implementations can achieve different levels of consistency, including row-level consistency, eventual consistency, or cross-table transactional consistency.
  - TiCDC does not have an implementation of Kafka consumers, but only provides **TiCDC Open Protocol**. You can implement the Kafka consumer according to this protocol.

### 10.10.1.3 Restrictions

There are some restrictions when using TiCDC.

#### 10.10.1.3.1 Requirements for valid index

TiCDC only replicates the table that has at least one **valid index**. A **valid index** is defined as follows:

- The primary key (`PRIMARY KEY`) is a valid index.
- The unique index (`UNIQUE INDEX`) that meets the following conditions at the same time is a valid index:
  - Every column of the index is explicitly defined as non-nullable (`NOT NULL`).
  - The index does not have the virtual generated column (`VIRTUAL GENERATED` ↞ `COLUMNS`).

Since v4.0.8, TiCDC supports replicating tables **without a valid index** by modifying the task configuration. However, this compromises the guarantee of data consistency to some extent. For more details, see [Replicate tables without a valid index](#).

#### 10.10.1.3.2 Unsupported scenarios

Currently, the following scenarios are not supported:

- The TiKV cluster that uses RawKV alone.
- The DDL operation `CREATE SEQUENCE` and the `SEQUENCE` function in TiDB. When the upstream TiDB uses `SEQUENCE`, TiCDC ignores `SEQUENCE` DDL operations/functions performed upstream. However, DML operations using `SEQUENCE` functions can be correctly replicated.

TiCDC only provides partial support for scenarios of large transactions in the upstream. For details, refer to [FAQ: Does TiCDC support replicating large transactions? Is there any risk?](#).

**Note:**

Since v5.3.0, TiCDC no longer supports the cyclic replication feature.

#### 10.10.1.4 Install and deploy TiCDC

You can either deploy TiCDC along with a new TiDB cluster or add the TiCDC component to an existing TiDB cluster. For details, see [Deploy TiCDC](#).

#### 10.10.1.5 Manage TiCDC Cluster and Replication Tasks

Currently, you can use the `cdc cli` tool to manage the status of a TiCDC cluster and data replication tasks. For details, see:

- Use `cdc cli` to manage cluster status and data replication task
- Use OpenAPI to manage cluster status and data replication task

#### 10.10.1.6 TiCDC Open Protocol

TiCDC Open Protocol is a row-level data change notification protocol that provides data sources for monitoring, caching, full-text indexing, analysis engines, and primary-secondary replication between different databases. TiCDC complies with TiCDC Open Protocol and replicates data changes of TiDB to third-party data medium such as MQ (Message Queue). For more information, see [TiCDC Open Protocol](#).

#### 10.10.1.7 Compatibility notes

#### 10.10.1.7.1 Incompatibility issue caused by using the TiCDC v5.0.0-rc cdc cli tool to operate a v4.0.x cluster

When using the `cdc cli` tool of TiCDC v5.0.0-rc to operate a v4.0.x TiCDC cluster, you might encounter the following abnormal situations:

- If the TiCDC cluster is v4.0.8 or an earlier version, using the v5.0.0-rc `cdc cli` tool to create a replication task might cause cluster anomalies and get the replication task stuck.
- If the TiCDC cluster is v4.0.9 or a later version, using the v5.0.0-rc `cdc cli` tool to create a replication task will cause the old value and unified sorter features to be unexpectedly enabled by default.

Solutions: Use the `cdc` executable file corresponding to the TiCDC cluster version to perform the following operations:

1. Delete the changefeed created using the v5.0.0-rc `cdc cli` tool. For example, run the `tiup cdc:v4.0.9 cli changefeed remove -c xxxx --pd=xxxxx --force` command.
2. If the replication task is stuck, restart the TiCDC cluster. For example, run the `tiup ↵ cluster restart <cluster_name> -R cdc` command.
3. Re-create the changefeed. For example, run the `tiup cdc:v4.0.9 cli changefeed ↵ create --sink-uri=xxxx --pd=xxx` command.

##### Note:

The above issue exists only when `cdc cli` is v5.0.0-rc. Other v5.0.x `cdc cli` tool can be compatible with v4.0.x clusters.

#### 10.10.1.7.2 Compatibility notes for `sort-dir` and `data-dir`

The `sort-dir` configuration is used to specify the temporary file directory for the TiCDC sorter. Its functionalities might vary in different versions. The following table lists `sort-dir`'s compatibility changes across versions.



Version	ality	Note	Recommendation
	sort		
	→ -		
	→ engine		
	→		
	func-		
	tion-		
v4.0.11	It is a or an ear- lier	In change- feed con- figu- ra- tion, item and speci- fies tem- po- rary file direc- tory for the file sorter and unified → sorter.	It these is not sions, rec- om- mended sorter to use <b>unified</b> sorter sorter in the pro- duc- tion <b>men-</b> <b>tal</b> <b>fea-</b> <b>tures</b> and <b>NOT</b> rec- dm- mended for the pro- duc- tion en- vi-
v4.0			
v5.0.0- rc			
1177		ron- ment. If mul-	

---

sort

↪ -

↪ engine

↪

func-

tion-

Version	ality	Note	Recommendation
---------	-------	------	----------------

v4.0.12, It is v4.0.13, a v5.0.0, con- and figu- v5.0.1 ra-	By de- fault, to <b>sort</b> fig- tion item of change- feed or of cdc ↪ <b>server</b> ↪ . a change- feed does not take ef- fect, and the <b>sort</b> TiUP).	You need the con- ure ↪ <b>disort</b> ↪ - ↪ - ↪ dir ↪ us- ing the <b>cdc</b> ↪ ↪ → server ↪ → com- fig- u- ra- tion of <b>cdc</b> ↪ ↪ <b>server</b> ↪ de-	
---	---	--	--

---

	sort	Version	ality	Note	Recommendation
v4.0.14	sort	You and	→ - can need		
		later	→ dir	con-	to
v4.0			fig-	con-	
ver-	is	ure	fig-		
	sions,	dep-	data	ure	
v5.0.3	re-	→ -	data		
	and	cated.	→ dir	→ -	
	later	It is	→	→ dir	
v5.0	rec-	us-	→		
ver-	om-	ing	us-		
	sions,	mended	the	ing	
	later	to	lat-	the	
TiDB	con-	est	cdc		
ver-	fig-	ver-	→		
	sions	ure	sion	→ server	
		data	of	→	
	→ -	TiUP.command-			
	→ dir	In	line		
	→ .	these pa-			
		TiDB ram-			
	ver-	e-			
	sions,	ter			
	unified				
	→	TiUP).			
		sorter			
		is			
		en-			
		abled			
		by			
		de-			
		fault.			
		Make			
		sure			
		that			
		data			
	→ -				
	→ dir				
	→				
1179	has				
	been				
	con-				
	g				

---

sort			
↳ -			
↳ engine			
↳			
func-			
tion-			
Version	alitry	Note	Recommendation

---

#### 10.10.1.7.3 Compatibility with temporary tables

Since v5.3.0, TiCDC supports [global temporary tables](#). Replicating global temporary tables to the downstream using TiCDC of a version earlier than v5.3.0 causes table definition error.

If the upstream cluster contains a global temporary table, the downstream TiDB cluster is expected to be v5.3.0 or a later version. Otherwise, an error occurs during the replication process.

#### 10.10.1.8 Troubleshoot TiCDC

For details, refer to [Troubleshoot TiCDC](#).

### 10.10.2 Deploy TiCDC

This document describes how to deploy a TiCDC cluster and the hardware and software recommendations for deploying and running it. You can either deploy TiCDC along with a new TiDB cluster or add the TiCDC component to an existing TiDB cluster. Generally, it is recommended that you deploy TiCDC using TiUP. In addition, you can also deploy it using binary as needed.

#### 10.10.2.1 Software and hardware recommendations

In production environments, the recommendations of software and hardware for TiCDC are as follows:

---

Linux OS	Version
Red Hat Enterprise Linux	7.3 or later versions
CentOS	7.3 or later versions

---

---

CPU	Memory	Disk type	Network	Number of TiCDC cluster instances (minimum required for production environment)
16 core+ 64 GB+	SSD	10 Gi-gabit net-work card (2 preferred )	2	

---

For more information, see [Software and Hardware Recommendations](#).

#### 10.10.2.2 Deploy a new TiDB cluster that includes TiCDC using TiUP

When you deploy a new TiDB cluster using TiUP, you can also deploy TiCDC at the same time. You only need to add the `cdc_servers` section in the initialization configuration file that TiUP uses to start the TiDB cluster. For detailed operations, see [Edit the initialization configuration file](#). For detailed configurable fields, see [Configure `cdc\_servers` using TiUP](#).

#### 10.10.2.3 Add TiCDC to an existing TiDB cluster using TiUP

You can also use TiUP to add the TiCDC component to an existing TiDB cluster. Take the following procedures:

1. Make sure that the current TiDB version supports TiCDC; otherwise, you need to upgrade the TiDB cluster to v4.0.0-rc.1 or later versions. Since v4.0.6, TiCDC has become a feature for general availability (GA). It is recommended that you use v4.0.6 or later versions.
2. To deploy TiCDC, refer to [Scale out a TiCDC cluster](#).

#### 10.10.2.4 Add TiCDC to an existing TiDB cluster using binary (not recommended)

Suppose that the PD cluster has a PD node (the client URL is 10.0.10.25:2379) that can provide services. If you want to deploy three TiCDC nodes, start the TiCDC cluster by executing the following commands. You only need to specify the same PD address, and the newly started nodes automatically join the TiCDC cluster.

```
cdc server --pd=http://10.0.10.25:2379 --log-file=ticdc_1.log --addr
    ↳ =0.0.0.0:8301 --advertise-addr=127.0.0.1:8301
cdc server --pd=http://10.0.10.25:2379 --log-file=ticdc_2.log --addr
    ↳ =0.0.0.0:8302 --advertise-addr=127.0.0.1:8302
cdc server --pd=http://10.0.10.25:2379 --log-file=ticdc_3.log --addr
    ↳ =0.0.0.0:8303 --advertise-addr=127.0.0.1:8303
```

#### 10.10.2.5 Description of TiCDC `cdc server` command-line parameters

The following are descriptions of options available in the `cdc server` command:

- **gc-ttl**: The TTL (Time To Live) of the service level GC safepoint in PD set by TiCDC, and the duration that the replication task can suspend, in seconds. The default value is 86400, which means 24 hours. Note: Suspending of the TiCDC replication task affects the progress of TiCDC GC safepoint, which means that it affects the progress of upstream TiDB GC, as detailed in [Complete Behavior of TiCDC GC safepoint](#).
- **pd**: The URL of the PD client.
- **addr**: The listening address of TiCDC, the HTTP API address, and the Prometheus address of the service.
- **advertise-addr**: The access address of TiCDC to the outside world.
- **tz**: Time zone used by the TiCDC service. TiCDC uses this time zone when it internally converts time data types such as `TIMESTAMP` or when it replicates data to the downstream. The default is the local time zone in which the process runs. If you specify `time-zone` (in `sink-uri`) and `tz` at the time, the internal TiCDC processes use the time zone specified by `tz`, and the sink uses the time zone specified by `time-zone` for replicating data to the downstream.

- **log-file**: The address of the running log of the TiCDC process. The default is `cdc.log`.
- **log-level**: The log level when the TiCDC process is running. The default is `info`.
- **ca**: The path of the CA certificate file used by TiCDC, in the PEM format (optional).
- **cert**: The path of the certificate file used by TiCDC, in the PEM format (optional).
- **key**: The path of the certificate key file used by TiCDC, in the PEM format (optional).
- **config**: The address of the configuration file that TiCDC uses (optional). This option is supported since TiCDC v5.0.0. This option can be used in the TiCDC deployment since TiUP v1.4.0.
- **data-dir**: Specifies the directory that TiCDC uses when it needs to use disk to store files. Unified Sorter uses this directory to store temporary files. Make sure there is enough space available on the device where this directory is located. If you are using TiUP, you can configure `data_dir` in the `cdb_servers` section, or directly use the default `data_dir` path in `global`.
- **sort-dir**: Specifies the temporary file directory used by the sorting engine. **Note that this option is not supported since TiDB v4.0.13, v5.0.3 and v5.1.0. Do not use it any more.**

### 10.10.3 Manage TiCDC Cluster and Replication Tasks

This document describes how to upgrade TiCDC cluster and modify the configuration of TiCDC cluster using TiUP, and how to manage the TiCDC cluster and replication tasks using the command-line tool `cdc cli`.

You can also use the HTTP interface (the TiCDC OpenAPI feature) to manage the TiCDC cluster and replication tasks. For details, see [TiCDC OpenAPI](#).

#### 10.10.3.1 Upgrade TiCDC using TiUP

This section introduces how to upgrade the TiCDC cluster using TiUP. In the following example, assume that you need to upgrade TiCDC and the entire TiDB cluster to v5.3.0.

```
tiup update --self && \
tiup update --all && \
tiup cluster upgrade <cluster-name> v5.3.0
```

##### 10.10.3.1.1 Notes for upgrade

- The `changefeed` configuration has changed in TiCDC v4.0.2. See [Compatibility notes for the configuration file](#) for details.
- If you encounter any issues, see [Upgrade TiDB using TiUP - FAQ](#).

### 10.10.3.2 Modify TiCDC configuration using TiUP

This section introduces how to modify the configuration of TiCDC cluster using the `tiup cluster edit-config` command of TiUP. The following example changes the value of `gc-ttl` from the default 86400 to 3600, namely, one hour.

First, execute the following command. You need to replace `<cluster-name>` with your actual cluster name.

```
tiup cluster edit-config <cluster-name>
```

Then, enter the vi editor page and modify the `cdc` configuration under `server-configs`. The configuration is shown below:

```
server_configs:
tidb: {}
tikv: {}
pd: {}
tiflash: {}
tiflash-learner: {}
pump: {}
drainer: {}
cdc:
  gc-ttl: 3600
```

After the modification, execute the `tiup cluster reload -R cdc` command to reload the configuration.

### 10.10.3.3 Use TLS

For details about using encrypted data transmission (TLS), see [Enable TLS Between TiDB Components](#).

### 10.10.3.4 Use `cdc cli` to manage cluster status and data replication task

This section introduces how to use `cdc cli` to manage a TiCDC cluster and data replication tasks. `cdc cli` is the `cli` sub-command executed using the `cdc` binary. The following interface description assumes that:

- `cli` commands are executed directly using the `cdc` binary;
- PD listens on 10.0.10.25 and the port is 2379.

**Note:**

The IP address and port that PD listens on correspond to the `advertise ↵ -client-urls` parameter specified during the `pd-server` startup.

Multiple pd-servers have multiple `advertise-client-urls` parameters and you can specify one or multiple parameters. For example, --  
 → `pd=http://10.0.10.25:2379` or `--pd=http://10.0.10.25:2379,http  
 → ://10.0.10.26:2379,http://10.0.10.27:2379.`

If you deploy TiCDC using TiUP, replace `cdc cli` in the following commands with `tiup ctl cdc`.

#### 10.10.3.4.1 Manage TiCDC service progress (`capture`)

- Query the `capture` list:

```
cdc cli capture list --pd=http://10.0.10.25:2379
```

```
[
  {
    "id": "806e3a1b-0e31-477f-9dd6-f3f2c570abdd",
    "is-owner": true,
    "address": "127.0.0.1:8300"
  },
  {
    "id": "ea2a4203-56fe-43a6-b442-7b295f458ebc",
    "is-owner": false,
    "address": "127.0.0.1:8301"
  }
]
```

- `id`: The ID of the service process.
- `is-owner`: Indicates whether the service process is the owner node.
- `address`: The address via which the service process provides interface to the outside.

#### 10.10.3.4.2 Manage replication tasks (`changefeed`)

State transfer of replication tasks

The state of a replication task represents the running status of the replication task. During the running of TiCDC, replication tasks might fail with errors, be manually paused, resumed, or reach the specified `TargetTs`. These behaviors can lead to the change of the replication task state. This section describes the states of TiCDC replication tasks and the transfer relationships between states.

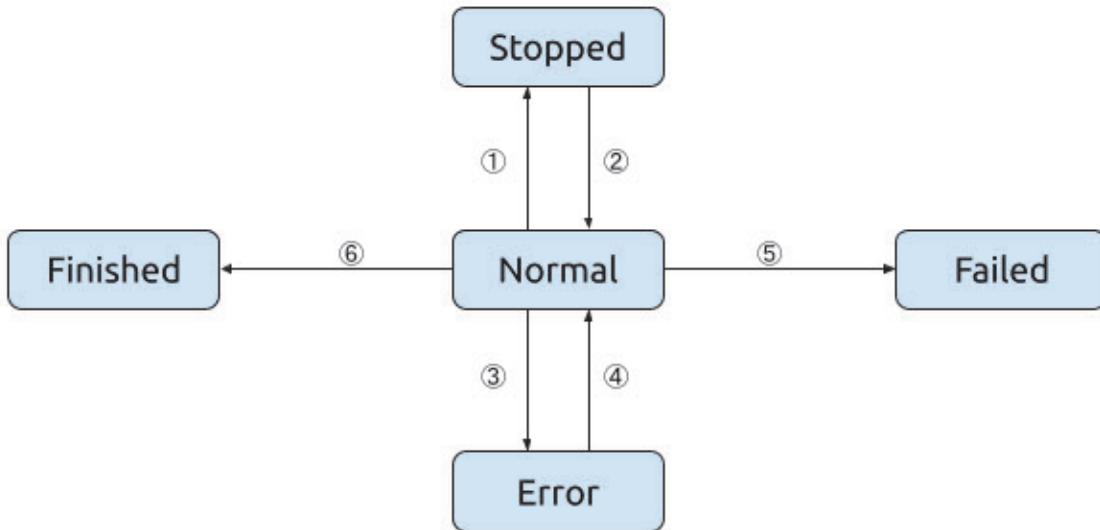


Figure 163: TiCDC state transfer

The states in the above state transfer diagram are described as follows:

- **Normal**: The replication task runs normally and the checkpoint-ts proceeds normally.
- **Stopped**: The replication task is stopped, because the user manually pauses the changefeed. The changefeed in this state blocks GC operations.
- **Error**: The replication task returns an error. The replication cannot continue due to some recoverable errors. The changefeed in this state keeps trying to resume until the state transfers to **Normal**. The changefeed in this state blocks GC operations.
- **Finished**: The replication task is finished and has reached the preset `TargetTs`. The changefeed in this state does not block GC operations.
- **Failed**: The replication task fails. Due to some unrecoverable errors, the replication task cannot resume and cannot be recovered. The changefeed in this state does not block GC operations.

The numbers in the above state transfer diagram are described as follows.

- Execute the `changefeed pause` command
- Execute the `changefeed resume` command to resume the replication task
- Recoverable errors occur during the `changefeed` operation, and the operation is resumed automatically.
- Execute the `changefeed resume` command to resume the replication task
- Recoverable errors occur during the `changefeed` operation
- `changefeed` has reached the preset `TargetTs`, and the replication is automatically stopped.

- **changefeed** suspended longer than the duration specified by `gc-ttl`, and cannot be resumed.
- **changefeed** experienced an unrecoverable error when trying to execute automatic recovery.

Create a replication task

Execute the following commands to create a replication task:

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
↪ root:123456@127.0.0.1:3306/" --changefeed-id="simple-replication-task"
↪ " --sort-engine="unified"
```

Create changefeed successfully!

```
ID: simple-replication-task
Info: {"sink-uri":"mysql://root:123456@127.0.0.1:3306/","opts":{},"create-
↪ time":"2020-03-12T22:04:08.103600025+08:00","start-ts
↪ ":415241823337054209,"target-ts":0,"admin-job-type":0,"sort-engine":"
↪ unified","sort-dir":".","config":{"case-sensitive":true,"filter":{"
↪ rules":["*.*"],"ignore-txn-start-ts":null,"ddl-allow-list":null},"
↪ mounter":{"worker-num":16},"sink":{"dispatchers":null,"protocol":"
↪ default"},"scheduler":{"type":"table-number","polling-time":-1}},"
↪ state":"normal","history":null,"error":null}
```

- **--changefeed-id**: The ID of the replication task. The format must match the `^[-a-zA-Z0-9]+([-a-zA-Z0-9]+)*$` regular expression. If this ID is not specified, TiCDC automatically generates a UUID (the version 4 format) as the ID.
- **--sink-uri**: The downstream address of the replication task. Configure `--sink-uri` according to the following format. Currently, the scheme supports `mysql/tidb/kafka` ↪ `/pulsar/s3/local`.

```
[scheme]://[[userinfo@] [host] : [port] [/path]?[query_parameters]]
```

When a URI contains special characters, you need to process these special characters using URL encoding.

- **--start-ts**: Specifies the starting TSO of the **changefeed**. From this TSO, the TiCDC cluster starts pulling data. The default value is the current time.
- **--target-ts**: Specifies the ending TSO of the **changefeed**. To this TSO, the TiCDC cluster stops pulling data. The default value is empty, which means that TiCDC does not automatically stop pulling data.
- **--sort-engine**: Specifies the sorting engine for the **changefeed**. Because TiDB and TiKV adopt distributed architectures, TiCDC must sort the data changes before writing them to the sink. This option supports `unified` (by default)/`memory/file`.

- **unified**: When **unified** is used, TiCDC prefers data sorting in memory. If the memory is insufficient, TiCDC automatically uses the disk to store the temporary data. This is the default value of **--sort-engine**.
  - **memory**: Sorts data changes in memory. It is **NOT recommended** to use this sorting engine, because OOM is easily triggered when you replicate a large amount of data.
  - **file**: Entirely uses the disk to store the temporary data. This feature is **deprecated**. It is **NOT recommended** to use it in **any** situation.
- **--config**: Specifies the configuration file of the `changefeed`.
  - **sort-dir**: Specifies the temporary file directory used by the sorting engine. **Note that this option is not supported since TiDB v4.0.13, v5.0.3 and v5.1.0. Do not use it any more.**

Configure sink URI with `mysql/tidb`

Sample configuration:

```
--sink-uri="mysql://root:123456@127.0.0.1:3306/?worker-count=16&max-txn-row
↪ =5000"
```

The following are descriptions of parameters and parameter values that can be configured for the sink URI with `mysql/tidb`:

Parameter/Parameter	
Value	Description
<code>root</code>	The username of the downstream database
<code>123456</code>	The password of the downstream database
<code>127.0.0.1</code>	The IP address of the downstream database
<code>3306</code>	The port for the downstream data
<code>worker-count</code>	The number of SQL statements that can be concurrently executed to the downstream (optional, 16 by default)
<code>max-txn-row</code>	The size of a transaction batch that can be executed to the downstream (optional, 256 by default)
<code>ssl-ca</code>	The path of the CA certificate file needed to connect to the downstream MySQL instance (optional)
<code>ssl-cert</code>	The path of the certificate file needed to connect to the downstream MySQL instance (optional)
<code>ssl-key</code>	The path of the certificate key file needed to connect to the downstream MySQL instance (optional)

---

Parameter/Parameter	
Value	Description
<code>time-zone</code>	The time zone used when connecting to the downstream MySQL instance, which is effective since v4.0.8. This is an optional parameter. If this parameter is not specified, the time zone of TiCDC service processes is used. If this parameter is set to an empty value, no time zone is specified when TiCDC connects to the downstream MySQL instance and the default time zone of the downstream is used.

---

Configure sink URI with `kafka`

Sample configuration:

```
--sink-uri="kafka://127.0.0.1:9092/topic-name?kafka-version=2.4.0&partition-
↪ num=6&max-message-bytes=67108864&replication-factor=1"
```

The following are descriptions of parameters and parameter values that can be configured for the sink URI with `kafka`:

---

Parameter/Parameter	
Value	Description
<code>127.0.0.1</code>	The IP address of the downstream Kafka services
<code>9092</code>	The port for the downstream Kafka
<code>topic-name</code>	Variable. The name of the Kafka topic
<code>kafka-version</code>	The version of the downstream Kafka. Optional, 2.4.0 by default. Currently, the earliest supported Kafka version is 0.11.0.2 and the latest one is 2.7.0. This value needs to be consistent with the actual version of the downstream Kafka.
<code>kafka-client-id</code>	Specifies the Kafka client ID of the replication task. Optional. <code>TiCDC_sarama_producer_replication_ID</code> by default.
<code>partition-num</code>	The number of the downstream Kafka partitions. Optional. The value must be <b>no greater than</b> the actual number of partitions. If you do not configure this parameter, the partition number is obtained automatically.
<code>max-message-bytes</code>	The maximum size of data that is sent to Kafka broker each time (optional, 64MB by default)
<code>replication- ↪ factor</code>	The number of Kafka message replicas that can be saved (optional, 1 by default)
<code>protocol</code>	The protocol with which messages are output to Kafka. The value options are <code>default</code> , <code>canal</code> , <code>avro</code> , and <code>maxwell</code> ( <code>default</code> by default)

---

Parameter/Parameter	Description
Value	Description
<b>max-batch-size</b>	New in v4.0.9. If the message protocol supports outputting multiple data changes to one Kafka message, this parameter specifies the maximum number of data changes in one Kafka message. It currently takes effect only when Kafka's <b>protocol</b> is <b>default</b> . (optional, 16 by default)
<b>ca</b>	The path of the CA certificate file needed to connect to the downstream Kafka instance (optional)
<b>cert</b>	The path of the certificate file needed to connect to the downstream Kafka instance (optional)
<b>key</b>	The path of the certificate key file needed to connect to the downstream Kafka instance (optional)
<b>sasl-user</b>	The identity (authcid) of SASL/PLAIN or SASL/SCRAM authentication needed to connect to the downstream Kafka instance (optional)
<b>sasl-password</b>	The password of SASL/PLAIN or SASL/SCRAM authentication needed to connect to the downstream Kafka instance (optional)
<b>sasl-mechanism</b>	The name of SASL/PLAIN or SASL/SCRAM authentication needed to connect to the downstream Kafka instance (optional)

---

**Note:**

When **protocol** is **default**, TiCDC tries to avoid generating messages that exceed **max-message-bytes** in length. However, if a row is so large that a single change alone exceeds **max-message-bytes** in length , to avoid silent failure, TiCDC tries to output this message and prints a warning in the log.

Integrate TiCDC with Kafka Connect (Confluent Platform)

**Warning:**

This is still an experimental feature. Do **NOT** use it in a production environment.

Sample configuration:

```
--sink-uri="kafka://127.0.0.1:9092/topic-name?kafka-version=2.4.0&protocol=
    ↪ avro&partition-num=6&max-message-bytes=67108864&replication-factor=1"
--opts registry="http://127.0.0.1:8081"
```

To use the [data connectors](#) provided by Confluent to stream data to relational or non-relational databases, you should use the `avro` protocol and provide a URL for [Confluent Schema Registry](#) in `opts`. Note that the `avro` protocol and Confluent integration are **experimental**.

For detailed integration guide, see [Quick Start Guide on Integrating TiDB with Confluent Platform](#).

Configure sink URI with `pulsar`

Sample configuration:

```
--sink-uri="pulsar://127.0.0.1:6650/topic-name?connectionTimeout=2s"
```

The following are descriptions of parameters that can be configured for the sink URI with `pulsar`:

Parameter	Description
<code>connectionTimeout</code>	The timeout for establishing a connection to the downstream Pulsar, which is optional and defaults to 30 (seconds)
<code>operationTimeout</code>	The timeout for performing an operation on the downstream Pulsar, which is optional and defaults to 30 (seconds)
<code>tlsTrustCertsFilePath</code>	The path of the CA certificate file needed to connect to the downstream Pulsar instance (optional)
<code>tlsAllowInsecureConnection</code>	Determines whether to allow unencrypted connection after TLS is enabled (optional)
<code>tlsValidateHostname</code>	Determines whether to verify the host name of the certificate from the downstream Pulsar (optional)
<code>maxConnectionsPerBroker</code>	The maximum number of connections allowed to a single downstream Pulsar broker, which is optional and defaults to 1
<code>auth.tls</code>	Uses the TLS mode to verify the downstream Pulsar (optional). For example, <code>auth=tls&amp;auth.tlsCertFile=/path/to/cert&amp;auth.tlsKeyFile=/path/to/key</code> .
<code>auth.token</code>	Uses the token mode to verify the downstream Pulsar (optional). For example, <code>auth=token&amp;auth.token=secret-token</code> or <code>auth=token&amp;auth.file=path/to/secret-token-file</code> .
<code>name</code>	The name of Pulsar producer in TiCDC (optional)
<code>maxPendingMessages</code>	Sets the maximum size of the pending message queue, which is optional and defaults to 1000. For example, pending for the confirmation message from Pulsar.
<code>disableBatching</code>	Disables automatically sending messages in batches (optional)
<code>batchingMaxPublishDelay</code>	Specifies the duration within which the messages sent are batched (default: 10ms)
<code>compressionType</code>	Sets the compression algorithm used for sending messages (optional). The value options are LZ4, ZLIB, and ZSTD (default).

Parameter	Description
<code>hashingScheme</code>	The hash algorithm used for choosing the partition to which a message is sent (optional). The value options are <code>JavaStringHash</code> (default) and <code>Murmur3</code> .
<code>properties.*</code>	The customized properties added to the Pulsar producer in TiCDC (optional). For example, <code>properties.location=Hangzhou</code> .

For more parameters of Pulsar, see [pulsar-client-go ClientOptions](#) and [pulsar-client-go ProducerOptions](#).

Use the task configuration file

For more replication configuration (for example, specify replicating a single table), see [Task configuration file](#).

You can use a configuration file to create a replication task in the following way:

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
↪ root:123456@127.0.0.1:3306/" --config changefeed.toml
```

In the command above, `changepool.toml` is the configuration file for the replication task.

Query the replication task list

Execute the following command to query the replication task list:

```
cdc cli changefeed list --pd=http://10.0.10.25:2379
```

```
[{
  "id": "simple-replication-task",
  "summary": {
    "state": "normal",
    "tso": 417886179132964865,
    "checkpoint": "2020-07-07 16:07:44.881",
    "error": null
  }
}]
```

- `checkpoint` indicates that TiCDC has already replicated data before this time point to the downstream.
- `state` indicates the state of the replication task.
  - `normal`: The replication task runs normally.
  - `stopped`: The replication task is stopped (manually paused).
  - `error`: The replication task is stopped (by an error).

- **removed**: The replication task is removed. Tasks of this state are displayed only when you have specified the `--all` option. To see these tasks when this option is not specified, execute the `changefeed query` command.
- **finished**: The replication task is finished (data is replicated to the `target-ts`). Tasks of this state are displayed only when you have specified the `--all` option. To see these tasks when this option is not specified, execute the `changefeed → query` command.

Query a specific replication task

To query a specific replication task, execute the `changefeed query` command. The query result includes the task information and the task state. You can specify the `--simple →` or `-s` argument to simplify the query result that will only include the basic replication state and the checkpoint information. If you do not specify this argument, detailed task configuration, replication states, and replication table information are output.

```
cdc cli changefeed query -s --pd=http://10.0.10.25:2379 --changefeed-id=
→ simple-replication-task
```

```
{
  "state": "normal",
  "tso": 419035700154597378,
  "checkpoint": "2020-08-27 10:12:19.579",
  "error": null
}
```

In the command and result above:

- `state` is the replication state of the current `changefeed`. Each state must be consistent with the state in `changefeed list`.
- `tso` represents the largest transaction TSO in the current `changefeed` that has been successfully replicated to the downstream.
- `checkpoint` represents the corresponding time of the largest transaction TSO in the current `changefeed` that has been successfully replicated to the downstream.
- `error` records whether an error has occurred in the current `changefeed`.

```
cdc cli changefeed query --pd=http://10.0.10.25:2379 --changefeed-id=simple-
→ replication-task
```

```
{
  "info": {
    "sink-uri": "mysql://127.0.0.1:3306/?max-txn-row=20\u0026worker-number
→ =4",
    "opts": {},
    "create-time": "2020-08-27T10:33:41.687983832+08:00",
```

```

"start-ts": 419036036249681921,
"target-ts": 0,
"admin-job-type": 0,
"sort-engine": "unified",
"sort-dir": ".",
"config": {
    "case-sensitive": true,
    "enable-old-value": false,
    "filter": {
        "rules": [
            "*.*"
        ],
        "ignore-txn-start-ts": null,
        "ddl-allow-list": null
    },
    "mounter": {
        "worker-num": 16
    },
    "sink": {
        "dispatchers": null,
        "protocol": "default"
    },
    "scheduler": {
        "type": "table-number",
        "polling-time": -1
    }
},
"state": "normal",
"history": null,
"error": null
},
"status": {
    "resolved-ts": 419036036249681921,
    "checkpoint-ts": 419036036249681921,
    "admin-job-type": 0
},
"count": 0,
"task-status": [
{
    "capture-id": "97173367-75dc-490c-ae2d-4e990f90da0f",
    "status": {
        "tables": {
            "47": {
                "start-ts": 419036036249681921
            }
        }
    }
}
]

```

```

    },
    "operation": null,
    "admin-job-type": 0
}
}
]
}

```

In the command and result above:

- **info** is the replication configuration of the queried **changefeed**.
- **status** is the replication state of the queried **changefeed**.
  - **resolved-ts**: The largest transaction TS in the current **changefeed**. Note that this TS has been successfully sent from TiKV to TiCDC.
  - **checkpoint-ts**: The largest transaction TS in the current **changefeed**. Note that this TS has been successfully written to the downstream.
  - **admin-job-type**: The status of a **changefeed**:
    - \* 0: The state is normal.
    - \* 1: The task is paused. When the task is paused, all replicated processors exit. The configuration and the replication status of the task are retained, so you can resume the task from **checkpiont-ts**.
    - \* 2: The task is resumed. The replication task resumes from **checkpoint-ts**.
    - \* 3: The task is removed. When the task is removed, all replicated processors are ended, and the configuration information of the replication task is cleared up. Only the replication status is retained for later queries.
- **task-status** indicates the state of each replication sub-task in the queried **changefeed**

Pause a replication task

Execute the following command to pause a replication task:

```
cdc cli changefeed pause --pd=http://10.0.10.25:2379 --changefeed-id simple-
→ replication-task
```

In the above command:

- **--changefeed-id=uuid** represents the ID of the **changefeed** that corresponds to the replication task you want to pause.

Resume a replication task

Execute the following command to resume a paused replication task:

```
cdc cli changefeed resume --pd=http://10.0.10.25:2379 --changefeed-id simple
↪ -replication-task
```

In the above command:

- `--changefeed-id=uuid` represents the ID of the `changefeed` that corresponds to the replication task you want to resume.

Remove a replication task

Execute the following command to remove a replication task:

```
cdc cli changefeed remove --pd=http://10.0.10.25:2379 --changefeed-id simple
↪ -replication-task
```

In the above command:

- `--changefeed-id=uuid` represents the ID of the `changefeed` that corresponds to the replication task you want to remove.

#### 10.10.3.4.3 Update task configuration

Starting from v4.0.4, TiCDC supports modifying the configuration of the replication task (not dynamically). To modify the `changefeed` configuration, pause the task, modify the configuration, and then resume the task.

```
cdc cli changefeed pause -c test-cf --pd=http://10.0.10.25:2379
cdc cli changefeed update -c test-cf --pd=http://10.0.10.25:2379 --sink-uri
↪ ="mysql://127.0.0.1:3306/?max-txn-row=20&worker-number=8" --config=
↪ changefeed.toml
cdc cli changefeed resume -c test-cf --pd=http://10.0.10.25:2379
```

Currently, you can modify the following configuration items:

- `sink-uri` of the `changefeed`.
- The `changefeed` configuration file and all configuration items in the file.
- Whether to use the file sorting feature and the sorting directory.
- The `target-ts` of the `changefeed`.

#### 10.10.3.4.4 Manage processing units of replication sub-tasks (processor)

- Query the `processor` list:

```
cdc cli processor list --pd=http://10.0.10.25:2379
```

```
[
  {
    "id": "9f84ff74-abf9-407f-a6e2-56aa35b33888",
    "capture-id": "b293999a-4168-4988-a4f4-35d9589b226b",
    "changefeed-id": "simple-replication-task"
  }
]
```

- Query a specific changefeed which corresponds to the status of a specific replication task:

```
cdc cli processor query --pd=http://10.0.10.25:2379 --changefeed-id=
  ↪ simple-replication-task --capture-id=b293999a-4168-4988-a4f4-35
  ↪ d9589b226b
```

```
{
  "status": {
    "tables": {
      "56": { # ID of the replication table, corresponding to
        ↪ tidb_table_id of a table in TiDB
        "start-ts": 417474117955485702
      }
    },
    "operation": null,
    "admin-job-type": 0
  },
  "position": {
    "checkpoint-ts": 417474143881789441,
    "resolved-ts": 417474143881789441,
    "count": 0
  }
}
```

In the command above:

- **status.tables**: Each key number represents the ID of the replication table, corresponding to `tidb_table_id` of a table in TiDB.
- **resolved-ts**: The largest TSO among the sorted data in the current processor.
- **checkpoint-ts**: The largest TSO that has been successfully written to the downstream in the current processor.

#### 10.10.3.5 Task configuration file

This section introduces the configuration of a replication task.

```

### Specifies whether the database names and tables in the configuration
    ↪ file are case-sensitive.
### The default value is true.
### This configuration item affects configurations related to filter and
    ↪ sink.
case-sensitive = true

### Specifies whether to output the old value. New in v4.0.5. Since v5.0,
    ↪ the default value is `true`.
enable-old-value = true

[filter]
### Ignores the transaction of specified start_ts.
ignore-txn-start-ts = [1, 2]

### Filter rules.
### Filter syntax: https://docs.pingcap.com/tidb/stable/table-filter#syntax.
rules = ['*.*', '!test.*']

[mounter]
### mounter thread counts, which is used to decode the TiKV output data.
worker-num = 16

[sink]
### For the sink of MQ type, you can use dispatchers to configure the event
    ↪ dispatcher.
### Supports four dispatchers: default, ts, rowid, and table.
### The dispatcher rules are as follows:
### - default: When multiple unique indexes (including the primary key)
    ↪ exist or the Old Value feature is enabled, events are dispatched in
    ↪ the table mode. When only one unique index (or the primary key)
    ↪ exists, events are dispatched in the rowid mode.
### - ts: Use the commitTs of the row change to create Hash and dispatch
    ↪ events.
### - rowid: Use the name and value of the selected HandleKey column to
    ↪ create Hash and dispatch events.
### - table: Use the schema name of the table and the table name to create
    ↪ Hash and dispatch events.
### The matching syntax of matcher is the same as the filter rule syntax.
dispatchers = [
    {matcher = ['test1.*', 'test2.*'], dispatcher = "ts"},
    {matcher = ['test3.*', 'test4.*'], dispatcher = "rowid"},
]
### For the sink of MQ type, you can specify the protocol format of the

```

```

    ↵ message.
### Currently four protocols are supported: default, canal, avro, and
    ↵ maxwell. The default protocol is TiCDC Open Protocol.
protocol = "default"

```

#### 10.10.3.5.1 Notes for compatibility

- In TiCDC v4.0.0, `ignore-txn-commit-ts` is removed and `ignore-txn-start-ts` is added, which uses `start_ts` to filter transactions.
- In TiCDC v4.0.2, `db-dbs/db-tables/ignore-dbs/ignore-tables` are removed and `rules` is added, which uses new filter rules for databases and tables. For detailed filter syntax, see [Table Filter](#).

#### 10.10.3.6 Output the historical value of a Row Changed Event New in v4.0.5

**Warning:**

Currently, outputting the historical value of a Row Changed Event is still an experimental feature. It is **NOT** recommended to use it in the production environment.

In the default configuration, the Row Changed Event of TiCDC Open Protocol output in a replication task only contains the changed value, not the value before the change. Therefore, the output value neither supports the [new collation framework](#) introduced in TiDB v4.0, nor can be used by the consumer ends of TiCDC Open Protocol as the historical value of a Row Changed Event.

Starting from v4.0.5, TiCDC supports outputting the historical value of a Row Changed Event. To enable this feature, specify the following configuration in the `changefeed` configuration file at the root level:

```
enable-old-value = true
```

After this feature is enabled, you can see [TiCDC Open Protocol - Row Changed Event](#) for the detailed output format. The new TiDB v4.0 collation framework will also be supported when you use the MySQL sink.

#### 10.10.3.7 Replicate tables without a valid index

Since v4.0.8, TiCDC supports replicating tables that have no valid index by modifying the task configuration. To enable this feature, configure in the `changefeed` configuration file as follows:

```
enable-old-value = true
force-replicate = true
```

**Warning:**

For tables without a valid index, operations such as `INSERT` and `REPLACE` are not reentrant, so there is a risk of data redundancy. TiCDC guarantees that data is distributed only at least once during the replication process. Therefore, enabling this feature to replicate tables without a valid index will definitely cause data redundancy. If you do not accept data redundancy, it is recommended to add an effective index, such as adding a primary key column with the `AUTO RANDOM` attribute.

#### 10.10.3.8 Unified Sorter

Unified sorter is the sorting engine in TiCDC. It can mitigate OOM problems caused by the following scenarios:

- The data replication task in TiCDC is paused for a long time, during which a large amount of incremental data is accumulated and needs to be replicated.
- The data replication task is started from an early timestamp so it becomes necessary to replicate a large amount of incremental data.

For the changefeeds created using `cdc cli` after v4.0.13, Unified Sorter is enabled by default; for the changefeeds that have existed before v4.0.13, the previous configuration is used.

To check whether or not the Unified Sorter feature is enabled on a changefeed, you can execute the following example command (assuming the IP address of the PD instance is `http://10.0.10.25:2379`):

```
cdc cli --pd="http://10.0.10.25:2379" changefeed query --changefeed-id=
→ simple-replication-task | grep 'sort-engine'
```

In the output of the above command, if the value of `sort-engine` is “unified”, it means that Unified Sorter is enabled on the changefeed.

**Note:**

- If your servers use mechanical hard drives or other storage devices that have high latency or limited bandwidth, use the unified sorter with caution.
- By default, Unified Sorter uses `data_dir` to store temporary files. It is recommended to ensure that the free disk space is greater than or equal to 500 GiB. For production environments, it is recommended to ensure that the free disk space on each node is greater than (the maximum `checkpoint-ts` delay allowed by the business) \* (upstream write traffic at business peak hours). In addition, if you plan to replicate a large amount of historical data after `changefeed` is created, make sure that the free space on each node is greater than the amount of replicated data.
- Unified sorter is enabled by default. If your servers do not match the above requirements and you want to disable the unified sorter, you need to manually set `sort-engine` to `memory` for the changefeed.
- To enable Unified Sorter on an existing changefeed that uses `memory` to sort, see the methods provided in [How do I handle the OOM that occurs after TiCDC is restarted after a task interruption?](#).

#### 10.10.3.9 Eventually consistent replication in disaster scenarios

Starting from v5.3.0, TiCDC provides the eventually consistent replication capability in disaster scenarios. When a disaster occurs in the primary TiDB cluster and the service cannot be resumed in a short period of time, TiCDC needs to provide the ability to ensure the consistency of data in the secondary cluster. Meanwhile, TiCDC needs to allow the business to quickly switch the traffic to the secondary cluster to avoid the database being unavailable for a long time and affecting the business.

This feature supports TiCDC to replicate incremental data from a TiDB cluster to the secondary relational database TiDB/Aurora/MySQL/MariaDB. In case the primary cluster crashes, TiCDC can recover the secondary cluster to a certain snapshot in the primary cluster within 5 minutes, given the condition that before disaster the replication status of TiCDC is normal and replication lag is small. It allows data loss of less than 30 minutes, that is, RTO <= 5min, and RPO <= 30min.

##### 10.10.3.9.1 Prerequisites

- Prepare a highly available Amazon S3 storage or NFS system for storing TiCDC's real-time incremental data backup files. These files can be accessed in case of an primary cluster disaster.
- Enable this feature for changefeeds that need to have eventual consistency in disaster scenarios. To enable it, you can add the following configuration to the changefeed configuration file.

```
[consistent]
### Consistency level. Options include:
### - none: the default value. In a non-disaster scenario, eventual
    ↪ consistency is only guaranteed if and only if finished-ts is
    ↪ specified.
### - eventual: Uses redo log to guarantee eventual consistency in case of
    ↪ the primary cluster disasters.
level = "eventual"

### Individual redo log file size, in MiB. By default, it's 64. It is
    ↪ recommended to be no more than 128.
max-log-size = 64

### The interval for flushing or uploading redo logs to S3, in milliseconds.
    ↪ By default, it's 1000. The recommended range is 500-2000.
flush-interval = 1000

### Form of storing redo log, including nfs (NFS directory) and S3 (
    ↪ uploading to S3).
storage = "s3://logbucket/test-changefeed?endpoint=http://$S3_ENDPOINT/"
```

#### 10.10.3.9.2 Disaster recovery

When a disaster happens in the primary cluster, you need to recover manually in the secondary cluster by running the `cdc redo` command. The recovery process is as follows.

1. Ensure that all the TiCDC processes have exited. This is to prevent the primary cluster from resuming service during data recovery and prevent TiCDC from restarting data synchronization.
2. Use cdc binary for data recovery. Run the following command:

```
cdc redo apply --tmp-dir="/tmp/cdc/redo/apply" \
--storage="s3://logbucket/test-changefeed?endpoint=http
    ↪ ://10.0.10.25:24927/" \
--sink-uri="mysql://normal:123456@10.0.10.55:3306/"
```

In this command:

- **tmp-dir**: Specifies the temporary directory for downloading TiCDC incremental data backup files.
- **storage**: Specifies the address for storing the TiCDC incremental data backup files, either an Amazon S3 storage or an NFS directory.
- **sink-uri**: Specifies the secondary cluster address to restore the data to. Scheme can only be `mysql`.

#### 10.10.4 Troubleshoot TiCDC

This document introduces the common issues and errors that you might encounter when using TiCDC, and the corresponding maintenance and troubleshooting methods.

##### Note:

In this document, the PD address specified in `cdc cli` commands is `--pd → =http://10.0.10.25:2379`. When you use the command, replace the address with your actual PD address.

##### 10.10.4.1 How do I choose `start-ts` when creating a task in TiCDC?

The `start-ts` of a replication task corresponds to a Timestamp Oracle (TSO) in the upstream TiDB cluster. TiCDC requests data from this TSO in a replication task. Therefore, the `start-ts` of the replication task must meet the following requirements:

- The value of `start-ts` is larger than the `tikv_gc_safe_point` value of the current TiDB cluster. Otherwise, an error occurs when you create a task.
- Before starting a task, ensure that the downstream has all data before `start-ts`. For scenarios such as replicating data to message queues, if the data consistency between upstream and downstream is not required, you can relax this requirement according to your application need.

If you do not specify `start-ts`, or specify `start-ts` as 0, when a replication task is started, TiCDC gets a current TSO and starts the task from this TSO.

##### 10.10.4.2 Why can't some tables be replicated when I create a task in TiCDC?

When you execute `cdc cli changefeed create` to create a replication task, TiCDC checks whether the upstream tables meet the [replication restrictions](#). If some tables do not meet the restrictions, `some tables are not eligible to replicate` is returned with a list of ineligible tables. You can choose Y or y to continue creating the task, and all updates on these tables are automatically ignored during the replication. If you choose an input other than Y or y, the replication task is not created.

##### 10.10.4.3 How do I view the state of TiCDC replication tasks?

To view the status of TiCDC replication tasks, use `cdc cli`. For example:

```
cdc cli changefeed list --pd=http://10.0.10.25:2379
```

The expected output is as follows:

```
[{
  "id": "4e24dde6-53c1-40b6-badf-63620e4940dc",
  "summary": {
    "state": "normal",
    "tso": 417886179132964865,
    "checkpoint": "2020-07-07 16:07:44.881",
    "error": null
  }
}]
```

- **checkpoint**: TiCDC has replicated all data before this timestamp to downstream.
- **state**: The state of this replication task:
  - **normal**: The task runs normally.
  - **stopped**: The task is stopped manually or encounters an error.
  - **removed**: The task is removed.

#### Note:

This feature is introduced in TiCDC 4.0.3.

### 10.10.4.4 TiCDC replication interruptions

#### 10.10.4.4.1 How do I know whether a TiCDC replication task is interrupted?

- Check the `changefeed checkpoint` monitoring metric of the replication task (choose the right `changefeed id`) in the Grafana dashboard. If the metric value stays unchanged, or the `checkpoint lag` metric keeps increasing, the replication task might be interrupted.
- Check the `exit error count` monitoring metric. If the metric value is greater than 0, an error has occurred in the replication task.
- Execute `cdc cli changefeed list` and `cdc cli changefeed query` to check the status of the replication task. `stopped` means the task has stopped, and the `error` item provides the detailed error message. After the error occurs, you can search `error ↴ on running processor` in the TiCDC server log to see the error stack for troubleshooting.
- In some extreme cases, the TiCDC service is restarted. You can search the `FATAL` level log in the TiCDC server log for troubleshooting.

#### 10.10.4.4.2 How do I know whether the replication task is stopped manually?

You can know whether the replication task is stopped manually by executing `cdc cli`. For example:

```
cdc cli changefeed query --pd=http://10.0.10.25:2379 --changefeed-id 28
↪ c43ffc-2316-4f4f-a70b-d1a7c59ba79f
```

In the output of the above command, `admin-job-type` shows the state of this replication task:

- 0: In progress, which means that the task is not stopped manually.
- 1: Paused. When the task is paused, all replicated processors exit. The configuration and the replication status of the task are retained, so you can resume the task from `checkpoint-ts`.
- 2: Resumed. The replication task resumes from `checkpoint-ts`.
- 3: Removed. When the task is removed, all replicated processors are ended, and the configuration information of the replication task is cleared up. The replication status is retained only for later queries.

#### 10.10.4.4.3 How do I handle replication interruptions?

A replication task might be interrupted in the following known scenarios:

- The downstream continues to be abnormal, and TiCDC still fails after many retries.
  - In this scenario, TiCDC saves the task information. Because TiCDC has set the service GC safepoint in PD, the data after the task checkpoint is not cleaned by TiKV GC within the valid period of `gc-ttl`.
    - Handling method: You can resume the replication task via the HTTP interface after the downstream is back to normal.
- Replication cannot continue because of incompatible SQL statement(s) in the downstream.
  - In this scenario, TiCDC saves the task information. Because TiCDC has set the service GC safepoint in PD, the data after the task checkpoint is not cleaned by TiKV GC within the valid period of `gc-ttl`.
    - Handling procedures:
      1. Query the status information of the replication task using the `cdc cli changefeed query` command and record the value of `checkpoint-ts`.
      2. Use the new task configuration file and add the `ignore-txn-start-ts` parameter to skip the transaction corresponding to the specified `start-ts`.

3. Stop the old replication task via HTTP API. Execute `cdc cli changefeed ↵ create` to create a new task and specify the new task configuration file. Specify `checkpoint-ts` recorded in step 1 as the `start-ts` and start a new task to resume the replication.
- In TiCDC v4.0.13 and earlier versions, when TiCDC replicates the partitioned table, it might encounter an error that leads to replication interruption.
    - In this scenario, TiCDC saves the task information. Because TiCDC has set the service GC safepoint in PD, the data after the task checkpoint is not cleaned by TiKV GC within the valid period of `gc-ttl`.
    - Handling procedures:
      1. Pause the replication task by executing `cdc cli changefeed pause -c < ↵ changefeed-id>`.
      2. Wait for about one munite, and then resume the replication task by executing `cdc cli changefeed resume -c <changefeed-id>`.

#### 10.10.4.4.4 What should I do to handle the OOM that occurs after TiCDC is restarted after a task interruption?

- Update your TiDB cluster and TiCDC cluster to the latest versions. The OOM problem has already been resolved in **v4.0.14 and later v4.0 versions, v5.0.2 and later v5.0 versions, and the latest versions**.
- In the above updated versions, you can enable the Unified Sorter to help you sort data in the disk when the system memory is insufficient. To enable this function, you can pass `--sort-engine=unified` to the `cdc cli` command when creating a replication task. For example:

```
cdc cli changefeed update -c <changefeed-id> --sort-engine="unified" --pd= ↵ http://10.0.10.25:2379
```

If you fail to update your cluster to the above new versions, you can still enable Unified Sorter in **previous versions**. You can pass `--sort-engine=unified` and `--sort-dir=/ ↵ path/to/sort_dir` to the `cdc cli` command when creating a replication task. For example:

```
cdc cli changefeed update -c <changefeed-id> --sort-engine="unified" --sort- ↵ dir="/data/cdc/sort" --pd=http://10.0.10.25:2379
```

**Note:**

- Since v4.0.9, TiCDC supports the unified sorter engine.
- TiCDC (the 4.0 version) does not support dynamically modifying the sorting engine yet. Make sure that the changefeed has stopped before modifying the sorter settings.
- `sort-dir` has different behaviors in different versions. Refer to [compatibility notes for `sort-dir` and `data-dir`](#), and configure it with caution.
- Currently, the unified sorter is an experimental feature. When the number of tables is too large ( $>=100$ ), the unified sorter might cause performance issues and affect replication throughput. Therefore, it is not recommended to use it in a production environment. Before you enable the unified sorter, make sure that the machine of each TiCDC node has enough disk capacity. If the total size of unprocessed data changes might exceed 1 TB, it is not recommended to use TiCDC for replication.

#### 10.10.4.5 What is `gc-ttl` in TiCDC?

Since v4.0.0-rc.1, PD supports external services in setting the service-level GC safepoint. Any service can register and update its GC safepoint. PD ensures that the key-value data later than this GC safepoint is not cleaned by GC.

When the replication task is unavailable or interrupted, this feature ensures that the data to be consumed by TiCDC is retained in TiKV without being cleaned by GC.

When starting the TiCDC server, you can specify the Time To Live (TTL) duration of GC safepoint by configuring `gc-ttl`. You can also [use TiUP to modify `gc-ttl`](#). The default value is 24 hours. In TiCDC, this value means:

- The maximum time the GC safepoint is retained at the PD after the TiCDC service is stopped.
- The maximum time a replication task can be suspended after the task is interrupted or manually stopped. If the time for a suspended replication task is longer than the value set by `gc-ttl`, the replication task enters the `failed` status, cannot be resumed, and cannot continue to affect the progress of the GC safepoint.

The second behavior above is introduced in TiCDC v4.0.13 and later versions. The purpose is to prevent a replication task in TiCDC from suspending for too long, causing the GC safepoint of the upstream TiKV cluster not to continue for a long time and retaining too many outdated data versions, thus affecting the performance of the upstream cluster.

**Note:**

In some scenarios, for example, when you use TiCDC for incremental replication after full replication with Dumpling/BR, the default 24 hours of `gc-ttl` may not be sufficient. You need to specify an appropriate value for `gc-ttl` when you start the TiCDC server.

#### 10.10.4.6 What is the complete behavior of TiCDC garbage collection (GC) safepoint?

If a replication task starts after the TiCDC service starts, the TiCDC owner updates the PD service GC safepoint with the smallest value of `checkpoint-ts` among all replication tasks. The service GC safepoint ensures that TiCDC does not delete data generated at that time and after that time. If the replication task is interrupted, or manually stopped, the `checkpoint-ts` of this task does not change. Meanwhile, PD's corresponding service GC safepoint is not updated either.

If the replication task is suspended longer than the time specified by `gc-ttl`, the replication task enters the `failed` status and cannot be resumed. The PD corresponding service GC safepoint will continue.

The Time-To-Live (TTL) that TiCDC sets for a service GC safepoint is 24 hours, which means that the GC mechanism does not delete any data if the TiCDC service can be recovered within 24 hours after it is interrupted.

#### 10.10.4.7 How do I handle the Error 1298: Unknown or incorrect time zone: 'UTC' error when creating the replication task or replicating data to MySQL?

This error is returned when the downstream MySQL does not load the time zone. You can load the time zone by running `mysql_tzinfo_to_sql`. After loading the time zone, you can create tasks and replicate data normally.

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql -p
```

If the output of the command above is similar to the following one, the import is successful:

```
Enter password:
Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone.
        → Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/leap-seconds.list' as time zone
        → . Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone.
        → Skipping it.
Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone.
        → Skipping it.
```

If the downstream is a special MySQL environment (a public cloud RDS or some MySQL derivative versions) and importing the time zone using the above method fails, you need to specify the MySQL time zone of the downstream using the `time-zone` parameter in `sink-uri`. You can first query the time zone used by MySQL:

1. Query the time zone used by MySQL:

```
show variables like '%time_zone%';
```

Variable_name	Value
system_time_zone	CST
time_zone	SYSTEM

2. Specify the time zone when you create the replication task and create the TiCDC service:

```
cdc cli changefeed create --sink-uri="mysql://root@127.0.0.1:3306/?time
→ -zone=CST" --pd=http://10.0.10.25:2379
```

#### Note:

CST might be an abbreviation for the following four different time zones:

- Central Standard Time (USA) UT-6:00
- Central Standard Time (Australia) UT+9:30
- China Standard Time UT+8:00
- Cuba Standard Time UT-4:00

In China, CST usually stands for China Standard Time.

#### 10.10.4.8 How to understand the relationship between the TiCDC time zone and the time zones of the upstream/downstream databases?

---

Upstre <i>TiCDC</i> Downstream	time time time	
zone zone zone		

---

ConfigSe <i>riation</i> Con <i>fig</i> Co <i>nfigured</i>		
method <i>Time</i> us- us-		
Zone ing ing		
Sup- the the		
port -- time		
↳ tz ↳ -		
↳     ↳ zone		
pa- ↳		
ram- pa-		
e- ram-		
ter e-		
when ter		
you in		
start sink		
the ↳ -		
TiCDC ↳ uri		
server ↳		

Upstream	TiDB	Downstream
time	time	time
zone	zone	zone
<b>Description:</b> TiCDC		
time	as	down-
zone	sumes	stream
of	that	MySQL
the	the	pro-
up-	up-	cesses
stream	stream	the
TiDB	TiDB	'times-
which	time	tamp
af-	zone	in
fects	is	the
DML	the	DML
op-	same	and
era-	as	DDL
tions	the	op-
of	TiCDC	era-
the	time	tions
times-zone	ac-	
tamp	cord-	
type	fig-	ing
and	u-	to
DDL	ra-	the
op-	tion,	down-
era-	and	stream
tions	per-	time
re-	forms	zone
lated	re-	set-
to	lated	ting.
times-op-		
tamp	era-	
type	tions	
column	on	
the		
times-		
tamp		
col-		
umn.		

**Note:**

Be careful when you set the time zone of the TiCDC server, because this time zone is used for converting the time type. Keep the upstream time zone, TiCDC time zone, and the downstream time zone consistent. The TiCDC server chooses its time zone in the following priority:

- TiCDC first uses the time zone specified using `--tz`.
- When `--tz` is not available, TiCDC tries to read the time zone set using the `TZ` environment variable.
- When the `TZ` environment variable is not available, TiCDC uses the default time zone of the machine.

#### 10.10.4.9 What is the default behavior of TiCDC if I create a replication task without specifying the configuration file in `--config`?

If you use the `cdc cli changefeed create` command without specifying the `-config` parameter, TiCDC creates the replication task in the following default behaviors:

- Replicates all tables except system tables
- Enables the Old Value feature
- Skips replicating tables that do not contain `valid indexes`

#### 10.10.4.10 How do I handle the incompatibility issue of configuration files caused by TiCDC upgrade?

Refer to [Notes for compatibility](#).

#### 10.10.4.11 Does TiCDC support outputting data changes in the Canal format?

Yes. To enable Canal output, specify the protocol as `canal` in the `--sink-uri` parameter. For example:

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="kafka
    ↳ ://127.0.0.1:9092/cdc-test?kafka-version=2.4.0&protocol=canal" --
    ↳ config changefeed.toml
```

#### Note:

- This feature is introduced in TiCDC 4.0.2.
- TiCDC currently supports outputting data changes in the Canal format only to MQ sinks such as Kafka and Pulsar.

For more information, refer to [Create a replication task](#).

#### 10.10.4.12 Why does the latency from TiCDC to Kafka become higher and higher?

- Check [how do I view the state of TiCDC replication tasks](#).
- Adjust the following parameters of Kafka:
  - Increase the `message.max.bytes` value in `server.properties` to 1073741824 (1 GB).
  - Increase the `replica.fetch.max.bytes` value in `server.properties` to 1073741824 (1 GB).
  - Increase the `fetch.message.max.bytes` value in `consumer.properties` to make it larger than the `message.max.bytes` value.

#### 10.10.4.13 When TiCDC replicates data to Kafka, does it write all the changes in a transaction into one message? If not, on what basis does it divide the changes?

No. According to the different distribution strategies configured, TiCDC divides the changes on different bases, including `default`, `row id`, `table`, and `ts`.

For more information, refer to [Replication task configuration file](#).

#### 10.10.4.14 When TiCDC replicates data to Kafka, can I control the maximum size of a single message in TiDB?

Yes. You can set the `max-message-bytes` parameter to control the maximum size of data sent to the Kafka broker each time (optional, 64MB by default). You can also set `max-<→ batch-size` to specify the maximum number of change records in each Kafka message. Currently, the setting only takes effect when Kafka's protocol is `default` (optional, 4096 by default).

#### 10.10.4.15 When TiCDC replicates data to Kafka, does a message contain multiple types of data changes?

Yes. A single message might contain multiple updates or deletes, and update and delete might co-exist.

#### 10.10.4.16 When TiCDC replicates data to Kafka, how do I view the timestamp, table name, and schema name in the output of TiCDC Open Protocol?

The information is included in the key of Kafka messages. For example:

```
{
  "ts":<TS>,
  "scm":<Schema Name>,
  "tbl":<Table Name>,
  "t":1
}
```

For more information, refer to [TiCDC Open Protocol event format](#).

#### 10.10.4.17 When TiCDC replicates data to Kafka, how do I know the timestamp of the data changes in a message?

You can get the unix timestamp by moving `ts` in the key of the Kafka message by 18 bits to the right.

#### 10.10.4.18 How does TiCDC Open Protocol represent null?

In TiCDC Open Protocol, the type code 6 represents `null`.

Type	Code	Output Example	Note
Null	6	{"t":6,"v":null}	

For more information, refer to [TiCDC Open Protocol column type code](#).

#### 10.10.4.19 The start-ts timestamp of the TiCDC task is quite different from the current time. During the execution of this task, replication is interrupted and an error [CDC:ErrBufferReachLimit] occurs

Since v4.0.9, you can try to enable the unified sorter feature in your replication task, or use the BR tool for an incremental backup and restore, and then start the TiCDC replication task from a new time.

#### 10.10.4.20 How can I tell if a Row Changed Event of TiCDC Open Protocol is an INSERT event or an UPDATE event?

If the Old Value feature is not enabled, you cannot tell whether a Row Changed Event of TiCDC Open Protocol is an `INSERT` event or an `UPDATE` event. If the feature is enabled, you can determine the event type by the fields it contains:

- `UPDATE` event contains both `"p"` and `"u"` fields
- `INSERT` event only contains the `"u"` field
- `DELETE` event only contains the `"d"` field

For more information, refer to [Open protocol Row Changed Event format](#).

#### 10.10.4.21 How much PD storage does TiCDC use?

TiCDC uses etcd in PD to store and regularly update the metadata. Because the time interval between the MVCC of etcd and PD's default compaction is one hour, the amount of PD storage that TiCDC uses is proportional to the amount of metadata versions generated within this hour. However, in v4.0.5, v4.0.6, and v4.0.7, TiCDC has a problem of frequent writing, so if there are 1000 tables created or scheduled in an hour, it then takes up all the etcd storage and returns the `etcdserver: mvcc: database space exceeded` error. You need to clean up the etcd storage after getting this error. See [etcd maintaince space-quota](#) for details. It is recommended to upgrade your cluster to v4.0.9 or later versions.

#### 10.10.4.22 Does TiCDC support replicating large transactions? Is there any risk?

TiCDC provides partial support for large transactions (more than 5 GB in size). Depending on different scenarios, the following risks might exist:

- When TiCDC's internal processing capacity is insufficient, the replication task error `ErrBufferReachLimit` might occur.
- When TiCDC's internal processing capacity is insufficient or the throughput capacity of TiCDC's downstream is insufficient, out of memory (OOM) might occur.

If you encounter an error above, it is recommended to use BR to restore the incremental data of large transactions. The detailed operations are as follows:

1. Record the `checkpoint-ts` of the changefeed that is terminated due to large transactions, use this TSO as the `--lastbackups` of the BR incremental backup, and execute [incremental data backup](#).
2. After backing up the incremental data, you can find a log record similar to `["Full backup Failed summary : total backup ranges: 0, total success: ↗ 0, total failed: 0"] [BackupTS=421758868510212097]` in the BR log output. Record the `BackupTS` in this log.
3. [Restore the incremental data](#).
4. Create a new changefeed and start the replication task from `BackupTS`.
5. Delete the old changefeed.

#### 10.10.4.23 When the downstream of a changefeed is a database similar to MySQL and TiCDC executes a time-consuming DDL statement, all other changefeeds are blocked. How should I handle the issue?

1. Pause the execution of the changefeed that contains the time-consuming DDL statement. Then you can see that other changefeeds are no longer blocked.
2. Search for the `apply job` field in the TiCDC log and confirm the `start-ts` of the time-consuming DDL statement.

3. Manually execute the DDL statement in the downstream. After the execution finishes, go on performing the following operations.
4. Modify the changefeed configuration and add the above `start-ts` to the `ignore-txn ↳ -start-ts` configuration item.
5. Resume the paused changefeed.

#### 10.10.4.24 After I upgrade the TiCDC cluster to v4.0.8, the [CDC:ErrKafkaInvalidConfig] Can requires old value to be enabled error is reported when I execute a changefeed

Since v4.0.8, if the `canal` or `maxwell` protocol is used for output in a changefeed, TiCDC enables the old value feature automatically. However, if you have upgraded TiCDC from an earlier version to v4.0.8 or later, when the changefeed uses the `canal` or `maxwell` protocol and the old value feature is disabled, this error is reported. To fix the error, take the following steps:

1. Set the value of `enable-old-value` in the changefeed configuration file to `true`.
2. Execute `cdc cli changefeed pause` to pause the replication task.

```
cdc cli changefeed pause -c test-cf --pd=http://10.0.10.25:2379
```

3. Execute `cdc cli changefeed update` to update the original changefeed configuration.

```
cdc cli changefeed update -c test-cf --pd=http://10.0.10.25:2379 --sink
    ↳ -uri="mysql://127.0.0.1:3306/?max-txn-row=20&worker-number=8" --
    ↳ config=changefeed.toml
```

4. Execute `cdc cli changefeed resume` to resume the replication task.

```
cdc cli changefeed resume -c test-cf --pd=http://10.0.10.25:2379
```

#### 10.10.4.25 The [tikv:9006] GC life time is shorter than transaction duration, transaction starts at xx, GC safe point is yy error is reported when I use TiCDC to create a changefeed

Solution: You need to execute the `pd-ctl service-gc-safepoint --pd <pd-addrs>` command to query the current GC safepoint and service GC safepoint. If the GC safepoint is smaller than the `start-ts` of the TiCDC replication task (changefeed), you can directly add the `--disable-gc-check` option to the `cdc cli create changefeed` command to create a changefeed.

If the result of `pd-ctl service-gc-safepoint --pd <pd-addrs>` does not have `gc_worker service_id`:

- If your PD version is v4.0.8 or earlier, refer to PD issue #3128 for details.

- If your PD is upgraded from v4.0.8 or an earlier version to a later version, refer to [PD issue #3366](#) for details.
- For other situations, report the execution result of the above command to our [Slack channel](#).

#### **10.10.4.26 enable-old-value is set to true when I create a TiCDC replication task, but INSERT/UPDATE statements from the upstream become REPLACE INTO after being replicated to the downstream**

When a changefeed is created in TiCDC, the `safe-mode` setting defaults to `true`, which generates the `REPLACE INTO` statement to execute for the upstream `INSERT/UPDATE` statements.

Currently, users cannot modify the `safe-mode` setting, so this issue currently has no solution.

#### **10.10.4.27 When I use TiCDC to replicate messages to Kafka, Kafka returns the Message was too large error**

For TiCDC v4.0.8 or earlier versions, you cannot effectively control the size of the message output to Kafka only by configuring the `max-message-bytes` setting for Kafka in the Sink URI. To control the message size, you also need to increase the limit on the bytes of messages to be received by Kafka. To add such a limit, add the following configuration to the Kafka server configuration.

```
### The maximum byte number of a message that the broker receives
message.max.bytes=2147483648
### The maximum byte number of a message that the broker copies
replica.fetch.max.bytes=2147483648
### The maximum message byte number that the consumer side reads
fetch.message.max.bytes=2147483648
```

#### **10.10.4.28 How can I find out whether a DDL statement fails to execute in downstream during TiCDC replication? How to resume the replication?**

If a DDL statement fails to execute, the replication task (changefeed) automatically stops. The `checkpoint-ts` is the DDL statement's `finish-ts` minus one. If you want TiCDC to retry executing this statement in the downstream, use `cdc cli changefeed resume` to resume the replication task. For example:

```
cdc cli changefeed resume -c test-cf --pd=http://10.0.10.25:2379
```

If you want to skip this DDL statement that goes wrong, set the `start-ts` of the changefeed to the `checkpoint-ts` (the timestamp at which the DDL statement goes wrong) plus one. For example, if the `checkpoint-ts` at which the DDL statement goes wrong is `415241823337054209`, execute the following commands to skip this DDL statement:

```
cdc cli changefeed update -c test-cf --pd=http://10.0.10.25:2379 --start-ts  
→ 415241823337054210  
cdc cli changefeed resume -c test-cf --pd=http://10.0.10.25:2379
```

10.10.4.29 The default value of the time type field is inconsistent when replicating a DDL statement to the downstream MySQL 5.7. What can I do?

Suppose that the `create table test (id int primary key, ts timestamp)` statement is executed in the upstream TiDB. When TiCDC replicates this statement to the downstream MySQL 5.7, MySQL uses the default configuration. The table schema after the replication is as follows. The default value of the `timestamp` field becomes `CURRENT_TIMESTAMP`:

```
mysql root@127.0.0.1:test> show create table test;
+---+
→ -----+
→
→
| Table | Create Table
→
→
+---+
→ -----+
→
→
| test | CREATE TABLE `test` (
→     |   `id` int(11) NOT NULL,
→     |   `ts` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
→         CURRENT_TIMESTAMP,
→     |   PRIMARY KEY (`id`)
→     | ) ENGINE=InnoDB DEFAULT CHARSET=latin1
→
+---+
→ -----+
→
```

From the result, you can see that the table schema before and after the replication is inconsistent. This is because the default value of `explicit_defaults_for_timestamp` in TiDB is different from that in MySQL. See [MySQL Compatibility](#) for details.

Since v5.0.1 or v4.0.13, for each replication to MySQL, TiCDC automatically sets `explicit_defaults_for_timestamp = ON` to ensure that the time type is consistent between the upstream and downstream. For versions earlier than v5.0.1 or v4.0.13, pay attention to the compatibility issue caused by the inconsistent `explicit_defaults_for_timestamp` value when using TiCDC to replicate the time type data.

#### 10.10.4.30 When the sink of the replication downstream is TiDB or MySQL, what permissions do users of the downstream database need?

When the sink is TiDB or MySQL, the users of the downstream database need the following permissions:

- Select
- Index
- Insert
- Update
- Delete
- Create
- Drop
- Alter
- Create View

If you need to replicate `recover` table to the downstream TiDB, the `Super` permission is required.

#### 10.10.5 Key Monitoring Metrics of TiCDC

If you use TiUP to deploy the TiDB cluster, you can see a sub-dashboard for TiCDC in the monitoring system which is deployed at the same time. You can get an overview of TiCDC's current status from the TiCDC dashboard, where the key metrics are displayed. This document provides a detailed description of these key metrics.

The metric description in this document is based on the following replication task example, which replicates data to MySQL using the default configuration.

```
cdc cli changefeed create --pd=http://10.0.10.25:2379 --sink-uri="mysql://
    ↪ root:123456@127.0.0.1:3306/" --changefeed-id="simple-replication-task
    ↪ "
```

The TiCDC dashboard contains four monitoring panels. See the following screenshot:

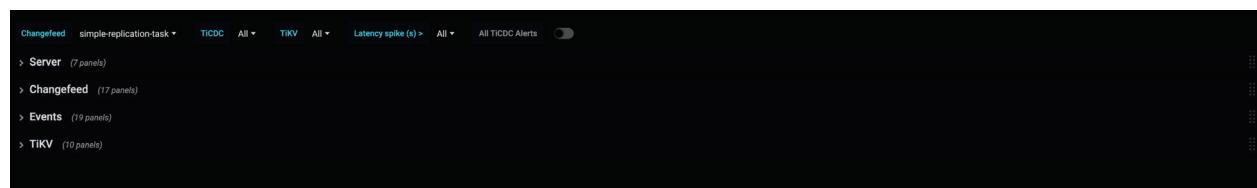


Figure 164: TiCDC Dashboard - Overview

The description of each panel is as follows:

- **Server:** The summary information of TiKV nodes and TiCDC nodes in the TiDB cluster
- **Changefeed:** The detailed information of TiCDC replication tasks
- **Events:** The detail information about the data flow within the TiCDC cluster
- **TiKV:** TiKV information related to TiCDC

#### 10.10.5.1 Server

The following is an example of the **Server** panel:



Figure 165: TiCDC Dashboard - Server metrics

The description of each metric in the **Server** panel is as follows:

- Uptime: The time for which TiKV nodes and TiCDC nodes have been running
- Goroutine count: The number of goroutines of a TiCDC node
- Open FD count: The number of file handles opened by TiCDC nodes
- Ownership: The current status of nodes in the TiCDC cluster
- Ownership history: The ownership history of the TiCDC cluster
- CPU usage: The CPU usage of TiCDC nodes
- Memory usage: The memory usage of TiCDC nodes

#### 10.10.5.2 Changefeed

The following is an example of the **Changefeed** panel:



The description of each metric in the **Changefeed** panel is as follows:

- **Changefeed table count:** The number of tables that each TiCDC node needs to replicate in the replication task
- **Processor resolved ts:** The timestamps that have been resolved in the TiCDC cluster
- **Table resolved ts:** The replication progress of each table in the replication task
- **Changefeed checkpoint:** The progress of replicating data to the downstream. Normally, the green bars are connected to the yellow line
- **PD etcd requests/s:** The number of requests that a TiCDC node sends to PD per second

- Exit error count: The number of errors that interrupt the replication task per minute
- Changefeed checkpoint lag: The progress lag of data replication (the unit is second) between the upstream and the downstream
- Changefeed resolved ts lag: The progress lag of data replication (the unit is second) between the upstream and TiCDC nodes
- Flush sink duration: The histogram of the time spent by TiCDC asynchronously flushing data to the downstream
- Flush sink duration percentile: The time (P95, P99, and P999) spent by TiCDC asynchronously flushing data to the downstream within one second
- Sink write duration: The histogram of the time spent by TiCDC writing a transaction change to the downstream
- Sink write duration percentile: The time (P95, P99, and P999) spent by TiCDC writing a transaction change to the downstream within one second
- MySQL sink conflict detect duration: The histogram of the time spent on detecting MySQL sink conflicts
- MySQL sink conflict detect duration percentile: The time (P95, P99, and P999) spent on detecting MySQL sink conflicts within one second
- MySQL sink worker load: The workload of MySQL sink workers of TiCDC nodes

#### 10.10.5.3 Events

The following is an example of the **Events** panel:





The description of each metric in the **Events** panel is as follows:

- Eventfeed count: The number of Eventfeed RPC requests of TiCDC nodes
- Event size percentile: The event size (P95, P99, and P999) which TiCDC receives from TiKV within one second
- Eventfeed error/m: The number of errors reported by Eventfeed RPC requests of TiCDC nodes per minute
- KV client receive events/s: The number of events that the KV client module of TiCDC nodes receives from TiKV per second
- Puller receive events/s: The number of events that the Puller module of TiCDC nodes receives from the KV client per second
- Puller output events/s: The number of events that the Puller module of TiCDC nodes sends to the Sorter module per second
- Sink flush rows/s: The number of events that TiCDC nodes write to the downstream per second
- Puller buffer size: The number of events that TiCDC nodes cache in the Puller module
- Entry sorter buffer size: The number of events that TiCDC nodes cache in the Sorter module
- Processor/Mounter buffer size: The number of events that TiCDC nodes cache in the Processor module and the Mounter module
- Sink row buffer size: The number of events that TiCDC nodes cache in the Sink module
- Entry sorter sort duration: The histogram of the time spent by TiCDC nodes sorting events
- Entry sorter sort duration percentile: The time (P95, P99, and P999) spent by TiCDC

sorting events within one second

- Entry sorter merge duration: The histogram of the time spent by TiCDC nodes merging sorted events
- Entry sorter merge duration percentile: The time (P95, P99, and P999) spent by TiCDC merging sorted events within one second
- Mounter unmarshal duration: The histogram of the time spent by TiCDC nodes unmarshaling events
- Mounter unmarshal duration percentile: The time (P95, P99, and P999) spent by TiCDC unmarshaling events within one second
- KV client dispatch events/s: The number of events that the KV client module dispatches among the TiCDC nodes
- KV client batch resolved size: The batch size of resolved timestamp messages that TiKV sends to TiCDC

#### 10.10.5.4 TiKV

The following is an example of the **TiKV** panel:



The description of each metric in the **TiKV** panel is as follows:

- CDC endpoint CPU: The CPU usage of the CDC endpoint threads on TiKV nodes
- CDC worker CPU: The CPU usage of the CDC worker threads on TiKV nodes
- Min resolved ts: The minimum resolved timestamp on TiKV nodes
- Min resolved region: The Region ID of the minimum resolved timestamp on TiKV nodes
- Resolved ts lag duration percentile: The lag between the minimum resolved timestamp on TiKV nodes and the current time
- Initial scan duration: The histogram of the time spent on incremental scan when TiKV nodes connect to TiCDC nodes
- Initial scan duration percentile: The time (P95, P99, and P999) spent on the incremental scan of TiKV nodes within one second
- Memory without block cache: The memory usage of TiKV nodes excluding the RocksDB block cache
- CDC pending bytes in memory: The memory usage of CDC module on TiKV nodes
- Captured region count: The number of event-capturing Regions on TiKV nodes

### 10.10.6 TiCDC Alert Rules

This document describes the TiCDC alert rules and the corresponding solutions. In descending order, the severity levels are: **Critical**, **Warning**.

#### 10.10.6.1 Critical alerts

This section introduces critical alerts and solutions.

##### 10.10.6.1.1 `cdc_checkpoint_high_delay`

For critical alerts, you need to pay close attention to abnormal monitoring metrics.

- Alert rule:  

$$(\text{time}() - \text{ticdc\_processor\_checkpoint\_ts} / 1000) > 600$$
- Description:  
A replication task is delayed more than 10 minutes.
- Solution:  
See [TiCDC Handle Replication Interruption](#).

#### 10.10.6.2 `cdc_resolvedts_high_delay`

- Alert rule:  

$$(\text{time}() - \text{ticdc\_processor\_resolved\_ts} / 1000) > 300$$

- Description:

The Resolved TS of a replication task is delayed more than 10 minutes.

- Solution:

See [TiCDC Handle Replication Interruption](#).

#### **10.10.6.2.1 ticdc\_processor\_exit\_with\_error\_count**

- Alert rule:

```
changes(ticdc_processor_exit_with_error_count[1m])> 0
```

- Description:

A replication task reports an error and exits.

- Solution:

See [TiCDC Handle Replication Interruption](#).

#### **10.10.6.3 Warning alerts**

Warning alerts are a reminder for an issue or error.

##### **10.10.6.3.1 cdc\_multiple\_owners**

- Alert rule:

```
sum(rate(ticdc_owner_ownership_counter[30s]))>= 2
```

- Description:

There are multiple owners in the TiCDC cluster.

- Solution:

Collect TiCDC logs to locate the root cause.

##### **10.10.6.3.2 ticdc\_mounter\_unmarshal\_and\_mount\_time\_more\_than\_1s**

- Alert rule:

```
histogram_quantile(0.9, rate(ticdc_mounter_unmarshal_and_mount_bucket[1m
˓→ ]))* 1000 > 1000
```

- Description:

It takes a replication task more than 1 second to unmarshal the data changes.

- Solution:

Collect TiCDC logs to locate the root cause.

#### 10.10.6.3.3 cdc\_sink\_execute\_duration\_time\_more\_than\_10s

- Alert rule:

```
histogram_quantile(0.9, rate(ticdc_sink_txn_exec_duration_bucket[1m]))>
    ↳ 10
```

- Description:

It takes a replication task more than 10 seconds to write data to the downstream database.

- Solution:

Check whether there are problems in the downstream database.

#### 10.10.6.3.4 cdc\_processor\_checkpoint\_tso\_no\_change\_for\_1m

- Alert rule:

```
changes(ticdc_processor_checkpoint_ts[1m]) < 1
```

- Description:

A replication task has not advanced for more than 1 minute.

- Solution:

See [TiCDC Handle Replication Interruption](#).

#### 10.10.6.3.5 ticdc\_puller\_entry\_sorter\_sort\_bucket

- Alert rule:

```
histogram_quantile(0.9, rate(ticdc_puller_entry_sorter_sort_bucket{}[1m
    ↳ ]) > 1
```

- Description:

The delay of TiCDC puller entry sorter is too high.

- Solution:

Collect TiCDC logs to locate the root cause.

#### 10.10.6.3.6 ticdc\_puller\_entry\_sorter\_merge\_bucket

- Alert rule:

```
histogram_quantile(0.9, rate(ticdc_puller_entry_sorter_merge_bucket{}[1m])) > 1
```

- Description:

The delay of TiCDC puller entry sorter merge is too high.

- Solution:

Collect TiCDC logs to locate the root cause.

#### 10.10.6.3.7 tikv\_cdc\_min\_resolved\_ts\_no\_change\_for\_1m

- Alert rule:

```
changes(tikv_cdc_min_resolved_ts[1m]) < 1 and ON (instance)tikv_cdc_region_resolve_status {"status": "resolved"} > 0
```

- Description:

The minimum Resolved TS 1 of TiKV CDC has not advanced for 1 minute.

- Solution:

Collect TiKV logs to locate the root cause.

#### 10.10.6.3.8 tikv\_cdc\_scan\_duration\_seconds\_more\_than\_10min

- Alert rule:

```
histogram_quantile(0.9, rate(tikv_cdc_scan_duration_seconds_bucket{}[1m])) > 600
```

- Description:

The TiKV CDC module has scanned for incremental replication for more than 10 minutes.

- Solution:

Collect TiCDC monitoring metrics and TiKV logs to locate the root cause.

#### 10.10.6.3.9 ticdc\_sink\_mysql\_execution\_error

- Alert rule:

```
changes(ticdc_sink_mysql_execution_error[1m]) > 0
```

- Description:

An error occurs when a replication task writes data to the downstream MySQL.

- Solution:

There are many possible root causes. See [Troubleshoot TiCDC](#).

#### 10.10.6.3.10 ticdc\_memory\_abnormal

- Alert rule:

```
go_memstats_heap_alloc_bytes{job="ticdc"} > 1e+10
```

- Description:

The TiCDC heap memory usage exceeds 10 GiB.

- Solution:

Collect TiCDC logs to locate the root cause.

### 10.10.7 TiCDC OpenAPI

TiCDC provides the OpenAPI feature for querying and operating the TiCDC cluster, which is similar to the feature of [cdc cli tool](#).

You can use the APIs to perform the following maintenance operations on the TiCDC cluster:

- Get the status information of a TiCDC node
- Check the health status of a TiCDC cluster
- Create a replication task
- Remove a replication task
- Update the replication configuration
- Query the replication task list
- Query a specific replication task
- Pause a replication task
- Resume a replication task
- Query the replication subtask list
- Query a specific replication subtask
- Query the TiCDC service process list
- Evict an owner node

- Manually trigger the load balancing of all tables in a replication task
- Manually schedule a table to another node
- Dynamically adjust the log level of the TiCDC server

The request body and returned value of all APIs are in JSON format. The following sections describe the specific usage of the APIs.

In the following examples, the listening IP address of the TiCDC server is 127.0.0.1 and the port is 8300. You can bind a specified IP and port via `--addr=ip:port` when starting the TiCDC server.

#### 10.10.7.1 API error message template

After sending an API request, if an error occurs, the returned error message is in the following format:

```
{
  "error_msg": "",
  "error_code": ""
}
```

From the above JSON output, `error_msg` describes the error message and `error_code` is the corresponding error code.

#### 10.10.7.2 Get the status information of a TiCDC node

This API is a synchronous interface. If the request is successful, the status information of the corresponding node is returned.

##### 10.10.7.2.1 Request URI

`GET /api/v1/status`

##### 10.10.7.2.2 Example

The following request gets the status information of the TiCDC node whose IP address is 127.0.0.1 and port number is 8300.

```
curl -X GET http://127.0.0.1:8300/api/v1/status
```

```
{
  "version": "v5.2.0-master-dirty",
  "git_hash": "f191cd00c53fdf7a2b1c9308a355092f9bf8824e",
  "id": "c6a43c16-0717-45af-afd6-8b3e01e44f5d",
  "pid": 25432,
  "is_owner": true
}
```

The fields of the above output are described as follows:

- version: The current TiCDC version number.
- git\_hash: The Git hash value.
- id: The capture ID of the node.
- pid: The capture process PID of the node.
- is\_owner: Indicates whether the node is an owner.

### 10.10.7.3 Check the health status of a TiCDC cluster

This API is a synchronous interface. If the cluster is healthy, 200 OK is returned.

#### 10.10.7.3.1 Request URI

GET /api/v1/health

#### 10.10.7.3.2 Example

```
curl -X GET http://127.0.0.1:8300/api/v1/health
```

### 10.10.7.4 Create a replication task

This API is an asynchronous interface. If the request is successful, 202 Accepted is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

#### 10.10.7.4.1 Request URI

POST /api/v1/changefeeds

#### 10.10.7.4.2 Parameter description

Compared to the optional parameters for creating a replication task using the `cli` command, the optional parameters for creating such task using the API are not as complete. This API supports the following parameters.

Parameters for the request body

Parameter name | Description |

changefeed_id	STRING type. The ID of the replication task. (Optional)
start_ts	UINT64 type. Specifies the start TSO of the changefeed. (Optional)
target_ts	UINT64 type. Specifies the target TSO of the changefeed. (Optional)
sink_uri	STRING type. The downstream address of the replication task. (Required)
force_replicate	BOOLEAN type. Determines whether to forcibly replicate the tables without unique indexes. (Optional)

`ignore_ineligible_table` | BOOLEAN type. Determines whether to ignore the tables that cannot be replicated. (Optional) |  
`filter_rules` | STRING type array. The rules for table schema filtering. (Optional) |  
`ignore_txn_start_ts` | UINT64 type array. Ignores the transaction of a specified start\_ts. (Optional) |  
`mounter_worker_num` | INT type. The mounter thread number. (Optional) |  
`sink_config` | The configuration parameters of sink. (Optional) |

The meaning and format of `changefeed_id`, `start_ts`, `target_ts`, and `sink_uri` are the same as those described in the [Use cdc cli to create a replication task](#) document. For the detailed description of these parameters, see this document. Note that when you specify the certificate path in `sink_uri`, make sure you have uploaded the corresponding certificate to the corresponding TiCDC server.

Some other parameters in the above table are described further as follows.

`force_replicate`: This parameter defaults to `false`. When it is specified as `true`, TiCDC tries to forcibly replicate tables that do not have a unique index.

`ignore_ineligible_table`: This parameter defaults to `false`. When it is specified as `true`, TiCDC ignores tables that cannot be replicated.

`filter_rules`: The rules for table schema filtering, such as `filter_rules = ['foo → *.*', 'bar*.*']`. For details, see the [Table Filter](#) document.

`ignore_txn_start_ts`: When this parameter is specified, the specified `start_ts` is ignored. For example, `ignore-txn-start-ts = [1, 2]`.

`mounter_worker_num`: The thread number of mounter. Mounter is used to decode the data output from TiKV. The default value is 16.

The configuration parameters of sink are as follows:

```
{
  "dispatchers": [
    {"matcher": ["test1.*", "test2.*"], "dispatcher": "ts"}, 
    {"matcher": ["test3.*", "test4.*"], "dispatcher": "rowid"} 
  ],
  "protocal": "default"
}
```

`dispatchers`: For the sink of MQ type, you can use dispatchers to configure the event dispatcher. Four dispatchers are supported: `default`, `ts`, `rowid`, and `table`. The dispatcher rules are as follows:

- `default`: When multiple unique indexes (including the primary key) exist or the Old Value feature is enabled, events are dispatched in the `table` mode. When only one unique index (or the primary key) exists, events are dispatched in the `rowid` mode.
- `ts`: Uses the commitTs of the row change to create the hash value and dispatch events.

- **rowid**: Uses the name and value of the selected HandleKey column to create the hash value and dispatch events.
- **table**: Uses the schema name of the table and the table name to create the hash value and dispatch events.

**matcher**: The matching syntax of matcher is the same as the filter rule syntax.

**protocol**: For the sink of MQ type, you can specify the protocol format of the message. Currently four protocols are supported: `default`, `canal`, `avro`, and `maxwell`. The default protocol is the TiCDC Open Protocol.

#### 10.10.7.4.3 Example

The following request creates a replication task with an ID of `test5` and a `sink_uri` of `blackhome://`.

```
curl -X POST -H "'Content-type':'application/json'" http://127.0.0.1:8300/
  ↳ api/v1/changefeeds -d '{"changefeed_id":"test5","sink_uri":"blackhole
  ↳ ://"}'
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

#### 10.10.7.5 Remove a replication task

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

##### 10.10.7.5.1 Request URI

`DELETE /api/v1/changefeeds/{changefeed_id}`

##### 10.10.7.5.2 Parameter description

Path parameters

---

Parameter name	Description
<code>changefeed_id</code>	The ID of the replication task (changefeed) to be removed.

---

##### 10.10.7.5.3 Example

The following request removes the replication task with the ID `test1`.

```
curl -X DELETE http://127.0.0.1:8300/api/v1/changefeeds/test1
```

If the request is successful, 202 Accepted is returned. If the request fails, an error message and error code are returned.

#### 10.10.7.6 Update the replication configuration

This API is an asynchronous interface. If the request is successful, 202 Accepted is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

To modify the changefeed configuration, follow the steps of `pause the replication task` → `modify the configuration` → `resume the replication task`.

##### 10.10.7.6.1 Request URI

```
PUT /api/v1/changefeeds/{changefeed_id}
```

##### 10.10.7.6.2 Parameter description

Path parameters

Parameter name	Description
<code>changefeed_id</code>	The ID of the replication task (changefeed) to be updated.

Parameters for the request body

Currently, only the following configuration can be modified via the API.

Parameter name | Description |

target_ts	UINT64 type. Specifies the target TSO of the changefeed. (Optional)
sink_uri	STRING type. The downstream address of the replication task. (Optional)
filter_rules	STRING type array. The rules for table schema filtering. (Optional)
ignore_txn_start_ts	UINT64 type array. Ignores the transaction of a specified start_ts. (Optional)
mounter_worker_num	INT type. The mounter thread number. (Optional)
sink_config	The configuration parameters of sink. (Optional)

The meanings of the above parameters are the same as those in the [Create a replication task](#) section. See that section for details.

##### 10.10.7.6.3 Example

The following request updates the `mounter_worker_num` of the replication task with the ID `test1` to 32.

```
curl -X PUT -H "'Content-type':'application/json'" http://127.0.0.1:8300/
  ↪ api/v1/changefeeds/test1 -d '{"mounter_worker_num":32}'
```

If the request is successful, 202 Accepted is returned. If the request fails, an error message and error code are returned.

#### 10.10.7.7 Query the replication task list

This API is a synchronous interface. If the request is successful, the basic information of all nodes in the TiCDC cluster is returned.

##### 10.10.7.7.1 Request URI

GET /api/v1/changefeeds

##### 10.10.7.7.2 Parameter description

Query parameters

Parameter name | Description |

:—— | :—— | :—— |

**state** | When this parameter is specified, the replication status information only of this state is returned.(Optional) |

The value options for **state** are **all**, **normal**, **stopped**, **error**, **failed**, and **finished**.

If this parameter is not specified, the basic information of replication tasks whose state is normal, stopped, or failed is returned by default.

##### 10.10.7.7.3 Example

The following request queries the basic information of all replication tasks whose state is **normal**.

```
curl -X GET http://127.0.0.1:8300/api/v1/changefeeds?state=normal
```

```
[  
  {  
    "id": "test1",  
    "state": "normal",  
    "checkpoint_tso": 426921294362574849,  
    "checkpoint_time": "2021-08-10 14:04:54.242",  
    "error": null  
  },  
  {  
    "id": "test2",  
    "state": "normal",  
    "checkpoint_tso": 426921294362574849,  
    "checkpoint_time": "2021-08-10 14:04:54.242",  
    "error": null  
  }]
```

[ ]

The fields in the returned result above are described as follows:

- id: The ID of the replication task.
- state: The current `state` of the replication task.
- checkpoint\_tso: The TSO representation of the current checkpoint of the replication task.
- checkpoint\_time: The formatted time representation of the current checkpoint of the replication task.
- error: The error information of the replication task.

#### 10.10.7.8 Query a specific replication task

This API is a synchronous interface. If the request is successful, the detailed information of the specified replication task is returned.

##### 10.10.7.8.1 Request URI

```
GET /api/v1/changefeeds/{changefeed_id}
```

##### 10.10.7.8.2 Parameter description

Path parameters

Parameter name	Description
<code>changefeed_id</code>	The ID of the replication task (changefeed) to be queried.

##### 10.10.7.8.3 Example

The following request queries the detailed information of the replication task with the ID `test1`.

```
curl -X GET http://127.0.0.1:8300/api/v1/changefeeds/test1
```

```
{
  "id": "test1",
  "sink_uri": "blackhole://",
  "create_time": "2021-08-10 11:41:30.642",
  "start_ts": 426919038970232833,
  "target_ts": 0,
  "checkpoint_tso": 426921014615867393,
  "checkpoint_time": "2021-08-10 13:47:07.093",
  "sort_engine": "unified",
  "state": "normal",
```

```

    "error": null,
    "error_history": null,
    "creator_version": "",
    "task_status": [
        {
            "capture_id": "d8924259-f52f-4dfb-97a9-c48d26395945",
            "table_ids": [
                63,
                65
            ],
            "table_operations": {}
        }
    ]
}

```

### 10.10.7.9 Pause a replication task

This API is an asynchronous interface. If the request is successful, 202 Accepted is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

#### 10.10.7.9.1 Request URI

POST /api/v1/changefeeds/{changefeed\_id}/pause

#### 10.10.7.9.2 Parameter description

Path parameters

---

Parameter name	Description
changefeed_id	The ID of the replication task (changefeed) to be paused.

---

#### 10.10.7.9.3 Example

The following request pauses the replication task with the ID `test1`.

```
curl -X POST http://127.0.0.1:8300/api/v1/changefeeds/test1/pause
```

If the request is successful, 202 Accepted is returned. If the request fails, an error message and error code are returned.

### 10.10.7.10 Resume a replication task

This API is an asynchronous interface. If the request is successful, 202 Accepted is returned. The returned result only means that the server agrees to run the command but

does not guarantee that the command will be run successfully.

#### 10.10.7.10.1 Request URI

```
POST /api/v1/changefeeds/{changefeed_id}/resume
```

#### 10.10.7.10.2 Parameter description

Path parameters

Parameter name	Description
changefeed_id	The ID of the replication task (changefeed) to be resumed.

#### 10.10.7.10.3 Example

The following request resumes the replication task with the ID `test1`.

```
curl -X POST http://127.0.0.1:8300/api/v1/changefeeds/test1/resume
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

#### 10.10.7.11 Query the replication subtask list

This API is a synchronous interface. If the request is successful, the basic information of all replication subtasks (processor) is returned.

#### 10.10.7.11.1 Request URI

```
GET /api/v1/processors
```

#### 10.10.7.11.2 Example

```
curl -X GET http://127.0.0.1:8300/api/v1/processors
```

```
[  
  {  
    "changefeed_id": "test1",  
    "capture_id": "561c3784-77f0-4863-ad52-65a3436db6af"  
  }  
]
```

#### 10.10.7.12 Query a specific replication subtask

This API is a synchronous interface. If the request is successful, the detailed information of the specified replication subtask (processor) is returned.

#### 10.10.7.12.1 Request URI

```
GET /api/v1/processors/{changefeed_id}/{capture_id}
```

#### 10.10.7.12.2 Parameter description

Path parameters

Parameter name	Description
changefeed_id	The changefeed ID of the replication subtask to be queried.
capture_id	The capture ID of the replication subtask to be queried.

#### 10.10.7.12.3 Example

The following request queries the detailed information of a subtask whose `changefeed_id` is `test` and `capture_id` is `561c3784-77f0-4863-ad52-65a3436db6af`. A subtask can be identified by `changefeed_id` and `capture_id`.

```
curl -X GET http://127.0.0.1:8300/api/v1/processors/test1/561c3784-77f0  
→ -4863-ad52-65a3436db6af
```

```
{  
  "checkpoint_ts": 426919123303006208,  
  "resolved_ts": 426919123369066496,  
  "table_ids": [  
    63,  
    65  
  ],  
  "error": null  
}
```

#### 10.10.7.13 Query the TiCDC service process list

This API is a synchronous interface. If the request is successful, the basic information of all replication processes (`captures`) is returned.

#### 10.10.7.13.1 Request URI

```
GET /api/v1/captures
```

#### 10.10.7.13.2 Example

```
curl -X GET http://127.0.0.1:8300/api/v1/captures
```

```
[  
  {  
    "id": "561c3784-77f0-4863-ad52-65a3436db6af",  
    "is_owner": true,  
    "address": "127.0.0.1:8300"  
  }  
]
```

#### 10.10.7.14 Evict an owner node

This API is an asynchronous interface. If the request is successful, 202 Accepted is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

##### 10.10.7.14.1 Request URI

POST /api/v1/owner/resign

##### 10.10.7.14.2 Example

The following request evicts the current owner node of TiCDC and triggers a new round of elections to generate a new owner node.

```
curl -X POST http://127.0.0.1:8300/api/v1/owner/resign
```

If the request is successful, 202 Accepted is returned. If the request fails, an error message and error code are returned.

#### 10.10.7.15 Manually trigger the load balancing of all tables in a replication task

This API is an asynchronous interface. If the request is successful, 202 Accepted is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

##### 10.10.7.15.1 Request URI

POST /api/v1/changefeed/{changefeed\_id}/tables/rebalance\_table

##### 10.10.7.15.2 Parameter description

Path parameters

Parameter name	Description
changefeed_id	The ID of the replication task (changefeed) to be scheduled.

### 10.10.7.15.3 Example

The following request triggers the load balancing of all tables in the changefeed with the ID `test1`.

```
curl -X POST http://127.0.0.1:8300/api/v1/changefeeds/test1/tables/
    ↪ rebalance_table
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

### 10.10.7.16 Manually schedule a table to another node

This API is an asynchronous interface. If the request is successful, `202 Accepted` is returned. The returned result only means that the server agrees to run the command but does not guarantee that the command will be run successfully.

#### 10.10.7.16.1 Request URI

`POST /api/v1/changefeeds/{changefeed_id}/tables/move_table`

#### 10.10.7.16.2 Parameter description

Path parameters

Parameter name	Description
<code>changefeed_id</code>	The ID of the replication task (changefeed) to be scheduled.

Parameters for the request body

Parameter name	Description
<code>target_capture_id</code>	The ID of the target capture.
<code>table_id</code>	The ID of the table to be scheduled.

#### 10.10.7.16.3 Example

The following request schedules the table with the ID `49` in the changefeed with the ID `test1` to the capture with the ID `6f19a6d9-0f8c-4dc9-b299-3ba7c0f216f5`.

```
curl -X POST -H "'Content-type':'application/json'" http://127.0.0.1:8300/
    ↪ api/v1/changefeeds/changefeed-test1/tables/move_table -d '{
    ↪ capture_id":"6f19a6d9-0f8c-4dc9-b299-3ba7c0f216f5","table_id":49}'
```

If the request is successful, `202 Accepted` is returned. If the request fails, an error message and error code are returned.

### 10.10.7.17 Dynamically adjust the log level of the TiCDC server

This API is a synchronous interface. If the request is successful, 202 OK is returned.

#### 10.10.7.17.1 Request URI

POST /api/v1/log

#### 10.10.7.17.2 Request parameters

Parameters for the request body

Parameter name	Description
log_level	The log level you want to set.

log\_level supports the log levels provided by zap: “debug”, “info”, “warn”, “error”, “dpanic”, “panic”, and “fatal”.

#### 10.10.7.17.3 Example

```
curl -X POST -H "'Content-type':'application/json'" http://127.0.0.1:8300/
↪ api/v1/log -d '{"log_level":"debug"}'
```

If the request is successful, 202 OK is returned. If the request fails, an error message and error code are returned.

## 10.10.8 TiCDC Open Protocol

TiCDC Open Protocol is a row-level data change notification protocol that provides data sources for monitoring, caching, full-text indexing, analysis engines, and primary-secondary replication between different databases. TiCDC complies with TiCDC Open Protocol and replicates data changes of TiDB to third-party data medium such as MQ (Message Queue).

TiCDC Open Protocol uses Event as the basic unit to replicate data change events to the downstream. The Event is divided into three categories:

- Row Changed Event: Represents the data change in a row. When a row is changed, this Event is sent and contains information about the changed row.
- DDL Event: Represents the DDL change. This Event is sent after a DDL statement is successfully executed in the upstream. The DDL Event is broadcasted to every MQ Partition.
- Resolved Event: Represents a special time point before which the Event received is complete.

### 10.10.8.1 Restrictions

- In most cases, the Row Changed Event of a version is sent only once, but in special situations such as node failure and network partition, the Row Changed Event of the same version might be sent multiple times.
- On the same table, the Row Changed Events of each version which is first sent are incremented in the order of timestamps (TS) in the Event stream.
- Resolved Events are periodically broadcasted to each MQ Partition. The Resolved Event means that any Event with a TS earlier than Resolved Event TS has been sent to the downstream.
- DDL Events are broadcasted to each MQ Partition.
- Multiple Row Changed Events of a row are sent to the same MQ Partition.

### 10.10.8.2 Message format

A Message contains one or more Events, arranged in the following format:

Key:

Offset(Byte)	0~7	8~15	16~(15+length1)	...	...
Parameter	Protocol version	Length1	Event Key1	LengthN	Event KeyN

Value:

Offset(Byte)	0~7	8~(7+length1)	...	...
Parameter	Length1	Event Value1	LengthN	Event ValueN

- LengthN represents the length of the Nth key/value.
- The length and protocol version are the big-endian `int64` type.
- The version of the current protocol is 1.

### 10.10.8.3 Event format

This section introduces the formats of Row Changed Event, DDL Event, and Resolved Event.

#### 10.10.8.3.1 Row Changed Event

- Key:

```
{
    "ts":<TS>,
    "scm":<Schema Name>,
    "tbl":<Table Name>,
```

```
{
    "t":1
}
```

Parameter	Type	Description
TS	Number	The timestamp of the transaction that causes the row change.
Schema Name	String	The name of the schema where the row is in.
Table Name	String	The name of the table where the row is in.

- **Value:**

Insert event. The newly added row data is output.

```
{
    "u": {
        <Column Name>: {
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        },
        <Column Name>: {
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        }
    }
}
```

Update event. The newly added row data (“u”) and the row data before the update (“p”) are output. The latter (“p”) is output only when the old value feature is enabled.

```
{
    "u": {
        <Column Name>: {
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        },
        <Column Name>: {
            "t":<Column Type>,
            "h":<Where Handle>,
            "f":<Flag>,
            "v":<Column Value>
        }
    }
}
```

```

        }
    },
    "p": {
        <Column Name>: {
            "t": <Column Type>,
            "h": <Where Handle>,
            "f": <Flag>,
            "v": <Column Value>
        },
        <Column Name>: {
            "t": <Column Type>,
            "h": <Where Handle>,
            "f": <Flag>,
            "v": <Column Value>
        }
    }
}

```

Delete event. The deleted row data is output. When the old value feature is enabled, the `Delete` event includes all the columns of the deleted row data; when this feature is disabled, the `Delete` event only includes the `HandleKey` column.

```

{
    "d": {
        <Column Name>: {
            "t": <Column Type>,
            "h": <Where Handle>,
            "f": <Flag>,
            "v": <Column Value>
        },
        <Column Name>: {
            "t": <Column Type>,
            "h": <Where Handle>,
            "f": <Flag>,
            "v": <Column Value>
        }
    }
}

```

Parameter	Type	Description
Column Name	String	The column name.
Column Type	Number	The column type. For details, see <a href="#">Column Type Code</a> .

Parameter	Type	Description
Where Handle	Boolean	Determines whether this column can be the filter condition of the <code>Where</code> clause. When this column is unique on the table, <code>Where Handle</code> is <code>true</code> .
Flag	Number	The bit flags of columns. For details, see <a href="#">Bit flags of columns</a> .
Column Value	Any	The Column value.

#### 10.10.8.3.2 DDL Event

- Key:

```
{
  "ts":<TS>,
  "scm":<Schema Name>,
  "tbl":<Table Name>,
  "t":2
}
```

Parameter	Type	Description
TS	Number	The timestamp of the transaction that performs the DDL change.
Schema Name	String	The schema name of the DDL change, which might be an empty string.
Table Name	String	The table name of the DDL change, which might be an empty string.

- Value:

```
{
  "q":<DDL Query>,
  "t":<DDL Type>
}
```

---

Parameter	Type	Description
DDL Query	String	DDL Query SQL
DDL Type	String	The DDL type. For details, see <a href="#">DDL Type Code</a> .

---

#### 10.10.8.3.3 Resolved Event

- **Key:**

```
{
  "ts":<TS>,
  "t":3
}
```

---

Parameter	Type	Description
TS	Number	The Resolved timestamp. Any TS earlier than this Event has been sent.

---

- **Value:** None

#### 10.10.8.4 Examples of the Event stream output

This section shows and displays the output logs of the Event stream.

Suppose that you execute the following SQL statement in the upstream and the MQ Partition number is 2:

```
CREATE TABLE test.t1(id int primary key, val varchar(16));
```

From the following Log 1 and Log 3, you can see that the DDL Event is broadcasted to all MQ Partitions, and that the Resolved Event is periodically broadcasted to each MQ Partition.

1. [partition=0] [key="{\"ts\":415508856908021766,\"scm\":\"test\", \"tbl\":"t1\", \"t\":2}"] [value="{\"q\":\"CREATE TABLE test.t1(id int primary key, val varchar(16))\", \"t\":3}"]
2. [partition=0] [key="{\"ts\":415508856908021766,\"t\":3}"] [value=]
3. [partition=1] [key="{\"ts\":415508856908021766,\"scm\":\"test\", \"tbl\":"t1\", \"t\":2}"] [value="{\"q\":\"CREATE TABLE test.t1(id int primary key, val varchar(16))\", \"t\":3}"]
4. [partition=1] [key="{\"ts\":415508856908021766,\"t\":3}"] [value=]

Execute the following SQL statements in the upstream:

```
BEGIN;
INSERT INTO test.t1(id, val) VALUES (1, 'aa');
```

```

INSERT INTO test.t1(id, val) VALUES (2, 'aa');
UPDATE test.t1 SET val = 'bb' WHERE id = 2;
INSERT INTO test.t1(id, val) VALUES (3, 'cc');
COMMIT;

```

- From the following Log 5 and Log 6, you can see that Row Changed Events on the same table might be sent to different partitions based on the primary key, but changes to the same row are sent to the same partition so that the downstream can easily process the Event concurrently.
- From Log 6, multiple changes to the same row in a transaction are only sent in one Row Changed Event.
- Log 8 is a repeated event of Log 7. Row Changed Event might be repeated, but the first Event of each version is sent orderly.

```

5. [partition=0] [key="\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":1},\"val\":{\"t\":15,\"v\":\"YWE=\"}}}]"
6. [partition=1] [key="\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":2},\"val\":{\"t\":15,\"v\":\"YmI=\"}}}]"
7. [partition=0] [key="\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":3},\"val\":{\"t\":15,\"v\":\"Y2M=\"}}}]"
8. [partition=0] [key="\"ts\":415508878783938562,\"scm\":\"test\",\"tbl
   ↪ \":\"t1\",\"t\":1}"] [value="{\"u\":{\"id\":{\"t\":3,\"h\":true,\"v
   ↪ \":3},\"val\":{\"t\":15,\"v\":\"Y2M=\"}}}]"

```

Execute the following SQL statements in the upstream:

```

BEGIN;
DELETE FROM test.t1 WHERE id = 1;
UPDATE test.t1 SET val = 'dd' WHERE id = 3;
UPDATE test.t1 SET id = 4, val = 'ee' WHERE id = 2;
COMMIT;

```

- Log 9 is the Row Changed Event of the Delete type. This type of Event only contains primary key columns or unique index columns.
- Log 13 and Log 14 are Resolved Events. The Resolved Event means that in this Partition, any events smaller than the Resolved TS (including Row Changed Event and DDL Event) have been sent.

```

9. [partition=0] [key={"ts":415508881418485761,"scm":"test","tbl
    ↵ ":"t1","t":1}][value={"d":{"id":{"t":3,"h":true,"v
    ↵ ":1}}}]
10. [partition=1] [key={"ts":415508881418485761,"scm":"test","tbl
    ↵ ":"t1","t":1}][value={"d":{"id":{"t":3,"h":true,"v
    ↵ ":2}}}]
11. [partition=0] [key={"ts":415508881418485761,"scm":"test","tbl
    ↵ ":"t1","t":1}][value={"u":{"id":{"t":3,"h":true,"v
    ↵ ":3}, "val":{"t":15, "v": "ZGQ="}}}]
12. [partition=0] [key={"ts":415508881418485761,"scm":"test","tbl
    ↵ ":"t1","t":1}][value={"u":{"id":{"t":3,"h":true,"v
    ↵ ":4}, "val":{"t":15, "v": "ZWU="}}}]
13. [partition=0] [key={"ts":415508881038376963,"t":3}][value=]
14. [partition=1] [key={"ts":415508881038376963,"t":3}][value=]

```

#### 10.10.8.5 Protocol parsing for consumers

Currently, TiCDC does not provide the standard parsing library for TiCDC Open Protocol, but the Golang version and Java version of parsing demonstrations are provided. You can refer to the data format provided in this document and the following demonstrations to implement the protocol parsing for consumers.

- [Golang demo](#)
- [Java demo](#)

#### 10.10.8.6 Column type code

Column Type Code represents the column data type of the Row Changed Event.

Type	Code	Output Exam- ple	Description
TINYINT/BOOLEAN	1	{"t":1,"v":1}	
SMALLINT	2	{"t":2,"v":1}	
INT	3	{"t":3,"v":123}	
FLOAT	4	{"t":4,"v":153.123}	
DOUBLE	5	{"t":5,"v":153.123}	
NULL	6	{"t":6,"v":null}	
TIMESTAMP	7	{"t":7,"v":"1973-12-30 15:30:00"}	
BIGINT	8	{"t":8,"v":123}	
MEDIUMINT	9	{"t":9,"v":123}	

Type	Code	Output Exam- ple	Description
DATE	10/14	{"t":10,"v":"2000-01-01"}	
TIME	11	{"t":11,"v":"23:59:59"}	
DATETIME	12	{"t":12,"v":"2015-12-20 23:58:58"}	
YEAR	13	{"t":13,"v":1970}	
VARCHAR/VARBINARY	13/253	{"t":15,"v":Ttest} / {"t":15,"v":is"\x89PNG\r\n\x1a\n"} en- coded in UTF- 8. When the up- stream type is VARBINARY, in- vis- ible char- ac- ters are es- caped.	
BIT	16	{"t":16,"v":81}	
JSON	245	{"t":245,"v":"{\\"key1\\":\\"value1\\"}"}	
DECIMAL	246	{"t":246,"v":"129012.1230000"}	
ENUM	247	{"t":247,"v":1}	
SET	248	{"t":248,"v":3}	

Type	Code	Output Example	Description
TINYTEXT/TINYBLOB	49	{"t":249,"v":\r\n"5rWL6K+VdGV4dA=="} value is en- coded in Base64.	
MEDIUMTEXT/MEDIUMBLOB	250	{"t":250,"v":\r\n"5rWL6K+VdGV4dA=="} value is en- coded in Base64.	
LONGTEXT/LONGBLOB	251	{"t":251,"v":\r\n"5rWL6K+VdGV4dA=="} value is en- coded in Base64.	
TEXT/BLOB	252	{"t":252,"v":\r\n"5rWL6K+VdGV4dA=="} value is en- coded in Base64.	

Type	Code	Output Example	Description
CHAR/BINARY	254	<pre>{"t":254,"v":test} {"t":254,"v":"\x89PNG\r\n\x1a\n"}</pre>	<p>en-coded in UTF-8.</p> <p>When the up-stream type is BI-NARY, invisible characters are escaped.</p>
GEOMETRY	255		Unsupported

#### 10.10.8.7 DDL Type Code

DDL Type Code represents the DDL statement type of the DDL Event.

Type	Code
Create Schema	1
Drop Schema	2
Create Table	3
Drop Table	4
Add Column	5
Drop Column	6
Add Index	7
Drop Index	8
Add Foreign Key	9

Type	Code
Drop Foreign Key	10
Truncate Table	11
Modify Column	12
Rebase Auto ID	13
Rename Table	14
Set Default Value	15
Shard RowID	16
Modify Table Comment	17
Rename Index	18
Add Table Partition	19
Drop Table Partition	20
Create View	21
Modify Table Charset And Collate	22
Truncate Table Partition	23
Drop View	24
Recover Table	25
Modify Schema Charset And Collate	26
Lock Table	27
Unlock Table	28
Repair Table	29
Set TiFlash Replica	30
Update TiFlash Replica Status	31
Add Primary Key	32
Drop Primary Key	33
Create Sequence	34
Alter Sequence	35
Drop Sequence	36

#### 10.10.8.8 Bit flags of columns

The bit flags represent specific attributes of columns.

Bit	Value	Name	Description
1	0x01	BinaryFlag	Whether the column is a binary-encoded column.
2	0x02	HandleKeyFlag	Whether the column is a Handle index column.
3	0x04	GeneratedColumnFlag	Whether the column is a generated column.
4	0x08	PrimaryKeyFlag	Whether the column is a primary key column.
5	0x10	UniqueKeyFlag	Whether the column is a unique index column.
6	0x20	MultipleKeyFlag	Whether the column is a composite index column.
7	0x40	NullableFlag	Whether the column is a nullable column.
8	0x80	UnsignedFlag	Whether the column is an unsigned column.

Example:

If the value of a column flag is 85, the column is a nullable column, a unique index column, a generated column, and a binary-encoded column.

```
85 == 0b_101_0101
== NullableFlag | UniqueKeyFlag | GeneratedColumnFlag | BinaryFlag
```

If the value of a column is 46, the column is a composite index column, a primary key column, a generated column, and a Handle key column.

```
46 == 0b_010_1110
== MultipleKeyFlag | PrimaryKeyFlag | GeneratedColumnFlag | HandleKeyFlag
```

#### Note:

- **BinaryFlag** is meaningful only when the column type is BLOB/TEXT (including TINYBLOB/TINYTEXT and BINARY/CHAR). When the upstream column is the BLOB type, the **BinaryFlag** value is set to 1. When the upstream column is the TEXT type, the **BinaryFlag** value is set to 0.
- To replicate a table from the upstream, TiCDC selects a **valid index** as the Handle index. The **HandleKeyFlag** value of the Handle index column is set to 1.

#### 10.10.9 Quick Start Guide on Integrating TiDB with Confluent Platform

This document introduces how to integrate TiDB to Confluent Platform using **TiCDC**.

#### Warning:

This is still an experimental feature. Do **NOT** use it in a production environment.

[Confluent Platform](#) is a data streaming platform with Apache Kafka at its core. With many official and third-party sink connectors, Confluent Platform enables you to easily connect stream sources to relational or non-relational databases.

To integrate TiDB with Confluent Platform, you can use the TiCDC component with the Avro protocol. TiCDC can stream data changes to Kafka in the format that Confluent Platform recognizes. For the detailed integration guide, see the following sections:

### 10.10.9.1 Prerequisites

#### Note:

In this tutorial, the [JDBC sink connector](#) is used to replicate TiDB data to a downstream relational database. To make it simple, [SQLite](#) is used here as an example.

- Make sure that Zookeeper, Kafka, and Schema Registry are properly installed. It is recommended that you follow the [Confluent Platform Quick Start Guide](#) to deploy a local test environment.
- Make sure that JDBC sink connector is installed by running the following command. The result should contain `jdbc-sink`.

```
confluent local services connect connector list
```

### 10.10.9.2 Integration procedures

1. Save the following configuration into `jdbc-sink-connector.json`:

```
{  
  "name": "jdbc-sink-connector",  
  "config": {  
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",  
    "tasks.max": "1",  
    "topics": "testdb_test",  
    "connection.url": "sqlite:test.db",  
    "connection.ds.pool.size": 5,  
    "table.name.format": "test",  
    "auto.create": true,  
    "auto.evolve": true  
  }  
}
```

2. Create an instance of the JDBC sink connector by running the following command (assuming Kafka is listening on 127.0.0.1:8083):

```
curl -X POST -H "Content-Type: application/json" -d @jdbc-sink-  
→ connector.json http://127.0.0.1:8083/connectors
```

3. Deploy TiCDC in one of the following ways. If TiCDC is already deployed, you can skip this step.

- Deploy a new TiDB cluster that includes TiCDC using TiUP
- Add TiCDC to an existing TiDB cluster using TiUP
- Add TiCDC to an existing TiDB cluster using binary (not recommended)

Make sure that your TiDB and TiCDC clusters are healthy before proceeding.

4. Create a `changefeed` by running the `cdc cli` command:

```
./cdc cli changefeed create --pd="http://127.0.0.1:2379" --sink-uri="
    ↳ kafka://127.0.0.1:9092/testdb_test?protocol=avro" --opts "
    ↳ registry=http://127.0.0.1:8081"
```

**Note:**

Make sure that PD, Kafka, and Schema Registry are running on their respective default ports.

#### 10.10.9.3 Test data replication

After TiDB is integrated with Confluent Platform, you can follow the example procedures below to test the data replication.

1. Create the `testdb` database in your TiDB cluster:

```
CREATE DATABASE IF NOT EXISTS testdb;
```

Create the `test` table in `testdb`:

```
USE testdb;
CREATE TABLE test (
    id INT PRIMARY KEY,
    v TEXT
);
```

**Note:**

If you need to change the database name or the table name, change `topics` in `jdbc-sink-connector.json` accordingly.

2. Insert data into TiDB:

```
INSERT INTO test (id, v) values (1, 'a');
INSERT INTO test (id, v) values (2, 'b');
INSERT INTO test (id, v) values (3, 'c');
INSERT INTO test (id, v) values (4, 'd');
```

3. Wait a moment for data to be replicated to the downstream. Then check the downstream for data:

```
sqlite3 test.db
sqlite> SELECT * from test;
```

### 10.10.10 TiCDC Glossary

This glossary provides TiCDC-related terms and definitions. These terms appears in TiCDC logs, monitoring metrics, configurations, and documents.

For TiDB-related terms and definitions, refer to [TiDB glossary](#).

#### 10.10.10.1 C

##### 10.10.10.1.1 Capture

A single TiCDC instance on which the replication task of the cluster runs. Multiple captures form a TiCDC cluster.

##### 10.10.10.1.2 Changed data

The data to be written to TiCDC from the upstream TiDB cluster, including the DML-caused data changes and the DDL-caused table schema changes.

##### 10.10.10.1.3 Changefeed

An incremental replication task in TiCDC, which outputs the data change logs of several tables in a TiDB cluster to the designated downstream.

#### 10.10.10.2 O

##### 10.10.10.2.1 Owner

A [capture](#) of a special role that manages the TiCDC cluster and schedules replication tasks of the cluster. An owner is elected by captures and there is at most one owner at any time.

#### 10.10.10.3 P

#### 10.10.10.3.1 Processor

TiCDC replication tasks allocate data tables on TiCDC instances, and the processor refers to the replication processing unit of these tables. Processor tasks include pulling, sorting, restoring, and distributing changed data.

### 10.11 Dumpling Overview

This document introduces the data export tool - [Dumpling](#). Dumpling exports data stored in TiDB/MySQL as SQL or CSV data files and can be used to make a logical full backup or export.

For backups of SST files (key-value pairs) or backups of incremental data that are not sensitive to latency, refer to [BR](#). For real-time backups of incremental data, refer to [TiCDC](#).

#### Note:

PingCAP previously maintained a fork of the [mydumper project](#) with enhancements specific to TiDB. This fork has since been replaced by [Dumpling](#), which has been rewritten in Go, and supports more optimizations that are specific to TiDB. It is strongly recommended that you use Dumpling instead of mydumper.

For the overview of Mydumper, refer to [v4.0 Mydumper documentation](#).

#### 10.11.1 Improvements of Dumpling compared with Mydumper

1. Support exporting data in multiple formats, including SQL and CSV
2. Support the [table-filter](#) feature, which makes it easier to filter data
3. Support exporting data to Amazon S3 cloud storage.
4. More optimizations are made for TiDB:
  - Support configuring the memory limit of a single TiDB SQL statement
  - Support automatic adjustment of TiDB GC time for TiDB v4.0.0 and above
  - Use TiDB's hidden column `_tidb_rowid` to optimize the performance of concurrent data export from a single table
  - For TiDB, you can set the value of `tidb_snapshot` to specify the time point of the data backup. This ensures the consistency of the backup, instead of using `FLUSH TABLES WITH READ LOCK` to ensure the consistency.

#### 10.11.2 Dumpling introduction

Dumpling is written in Go. The Github project is [pingcap/dumpling](#).

For detailed usage of Dumpling, use the `--help` option or refer to [Option list of Dumpling](#).

When using Dumpling, you need to execute the export command on a running cluster. This document assumes that there is a TiDB instance on the `127.0.0.1:4000` host and that this TiDB instance has a root user without a password.

You can get Dumpling using [TiUP](#) by running `tiup install dumpling`. Afterwards, you can use `tiup dumpling ...` to run Dumpling.

Dumpling is also included in the `tidb-toolkit` installation package and can be [download here](#).

### 10.11.3 Export data from TiDB/MySQL

#### 10.11.3.1 Required privileges

- SELECT
- RELOAD
- LOCK TABLES
- REPLICATION CLIENT
- PROCESS

#### 10.11.3.2 Export to SQL files

Dumpling exports data to SQL files by default. You can also export data to SQL files by adding the `--filetype sql` flag:

```
dumpling \
-u root \
-P 4000 \
-h 127.0.0.1 \
--filetype sql \
-t 8 \
-o /tmp/test \
-r 200000 \
-F 256MiB
```

In the command above:

- The `-h`, `-p`, and `-u` option respectively mean the address, the port, and the user. If a password is required for authentication, you can use `-p $YOUR_SECRET_PASSWORD` to pass the password to Dumpling.
- The `-o` option specifies the export directory of the storage, which supports a local file path or a [URL of an external storage](#).

- The `-t` option specifies the number of threads for the export. Increasing the number of threads improves the concurrency of Dumpling and the export speed, and also increases the database's memory consumption. Therefore, it is not recommended to set the number too large. Usually, it's less than 64.
- The `-r` option specifies the maximum number of rows in a single file. With this option specified, Dumpling enables the in-table concurrency to speed up the export and reduce the memory usage.
- The `-F` option is used to specify the maximum size of a single file (the unit here is MiB; inputs like `5GiB` or `8KB` are also acceptable). It is recommended to keep its value to 256 MiB or less if you plan to use TiDB Lightning to load this file into a TiDB instance.

**Note:**

If the size of a single exported table exceeds 10 GB, it is **strongly recommended to use** the `-r` and `-F` options.

### 10.11.3.3 Export to CSV files

If Dumpling exports data to CSV files (use `--filetype csv` to export to CSV files), you can also use `--sql <SQL>` to export the records selected by the specified SQL statement.

For example, you can export all records that match `id < 100` in `test.sbtest1` using the following command:

```
./dumpling \
-u root \
-P 4000 \
-h 127.0.0.1 \
-o /tmp/test \
--filetype csv \
--sql 'select * from `test`.`sbtest1` where id < 100'
```

**Note:**

- Currently, the `--sql` option can be used only for exporting to CSV files.
- Here you need to execute the `select * from <table-name> where id ↪ <100` statement on all tables to be exported. If some tables do not have specified fields, the export fails.
- Strings and keywords are not distinguished in CSV files. If the imported data is the Boolean type, you need to convert `true` and `false` to 1 and 0.

#### 10.11.3.4 Format of exported files

- `metadata`: The start time of the exported files and the position of the master binary log.

```
cat metadata
```

```
Started dump at: 2020-11-10 10:40:19
SHOW MASTER STATUS:
    Log: tidb-binlog
    Pos: 420747102018863124
Finished dump at: 2020-11-10 10:40:20
```

- `{schema}-schema-create.sql`: The SQL file used to create the schema

```
cat test-schema-create.sql
```

```
CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4 */;
```

- `{schema}.{table}-schema.sql`: The SQL file used to create the table

```
cat test.t1-schema.sql
```

```
CREATE TABLE `t1` (
    `id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin;
```

- `{schema}.{table}.{0001}.{sql|csv}`: The date source file

```
cat test.t1.0.sql
```

```
/*!40101 SET NAMES binary*/;
INSERT INTO `t1` VALUES
(1);
```

- `*-schema-view.sql`, `*-schema-trigger.sql`, `*-schema-post.sql`: Other exported files

#### 10.11.3.5 Export data to Amazon S3 cloud storage

Since v4.0.8, Dumpling supports exporting data to cloud storages. If you need to back up data to Amazon's S3 backend storage, you need to specify the S3 storage path in the `-o` parameter.

You need to create an S3 bucket in the specified region (see the [Amazon documentation - How do I create an S3 Bucket](#)). If you also need to create a folder in the bucket, see the [Amazon documentation - Creating a folder](#).

Pass `SecretKey` and `AccessKey` of the account with the permission to access the S3 backend storage to the Dumpling node as environment variables.

```
export AWS_ACCESS_KEY_ID=${AccessKey}
export AWS_SECRET_ACCESS_KEY=${SecretKey}
```

Dumpling also supports reading credential files from `~/.aws/credentials`. For more Dumpling configuration, see the configuration of [External storages](#).

When you back up data using Dumpling, explicitly specify the `--s3.region` parameter, which means the region of the S3 storage (for example, `ap-northeast-1`):

```
./dumpling \
-u root \
-P 4000 \
-h 127.0.0.1 \
-r 200000 \
-o "s3://${Bucket}/${Folder}" \
--s3.region "${region}"
```

### 10.11.3.6 Filter the exported data

#### 10.11.3.6.1 Use the `--where` option to filter data

By default, Dumpling exports all databases except system databases (including `mysql`, `sys`, `INFORMATION_SCHEMA`, `PERFORMANCE_SCHEMA`, `METRICS_SCHEMA`, and `INSPECTION_SCHEMA`). You can use `--where <SQL where expression>` to select the records to be exported.

```
./dumpling \
-u root \
-P 4000 \
-h 127.0.0.1 \
-o /tmp/test \
--where "id < 100"
```

The above command exports the data that matches `id < 100` from each table. Note that you cannot use the `--where` parameter together with `--sql`.

#### 10.11.3.6.2 Use the `--filter` option to filter data

Dumpling can filter specific databases or tables by specifying the table filter with the `--filter` option. The syntax of table filters is similar to that of `.gitignore`. For details, see [Table Filter](#).

```
./dumpling \
-u root \
```

```
-P 4000 \
-h 127.0.0.1 \
-o /tmp/test \
-r 200000 \
--filter "employees.*" \
--filter "*.WorkOrder"
```

The above command exports all the tables in the `employees` database and the `WorkOrder` tables in all databases.

#### 10.11.3.6.3 Use the `-B` or `-T` option to filter data

Dumpling can also export specific databases with the `-B` option or specific tables with the `-T` option.

**Note:**

- The `--filter` option and the `-T` option cannot be used at the same time.
- The `-T` option can only accept a complete form of inputs like `database ↵ -name.table-name`, and inputs with only the table name are not accepted. Example: Dumpling cannot recognize `-T WorkOrder`.

Examples:

- `-B employees` exports the `employees` database.
- `-T employees.WorkOrder` exports the `employees.WorkOrder` table.

#### 10.11.3.7 Improve export efficiency through concurrency

The exported file is stored in the `./export-<current local time>` directory by default. Commonly used options are as follows:

- The `-t` option specifies the number of threads for the export. Increasing the number of threads improves the concurrency of Dumpling and the export speed, and also increases the database's memory consumption. Therefore, it is not recommended to set the number too large.
- The `-r` option specifies the maximum number of records (or the number of rows in the database) for a single file. When it is enabled, Dumpling enables concurrency in the table to improve the speed of exporting large tables.

- The `--compress gzip` option can be used to compress the dump. This can help to speed up dumping of data if storage is the bottleneck or if storage capacity is a concern. The drawback of this is an increase in CPU usage. Each file is compressed individually.

With the above options specified, Dumpling can have a quicker speed of data export.

#### 10.11.3.8 Adjust Dumpling's data consistency options

##### Note:

In most scenarios, you do not need to adjust the default data consistency options of Dumpling (the default value is `auto`).

Dumpling uses the `--consistency <consistency level>` option to control the way in which data is exported for “consistency assurance”. When using `snapshot` for consistency, you can use the `--snapshot` option to specify the timestamp to be backed up. You can also use the following levels of consistency:

- `flush`: Use `FLUSH TABLES WITH READ LOCK` to temporarily interrupt the DML and DDL operations of the replica database, to ensure the global consistency of the backup connection, and to record the binlog position (POS) information. The lock is released after all backup connections start transactions. It is recommended to perform full backups during off-peak hours or on the MySQL replica database.
- `snapshot`: Get a consistent snapshot of the specified timestamp and export it.
- `lock`: Add read locks on all tables to be exported.
- `none`: No guarantee for consistency.
- `auto`: Use `flush` for MySQL and `snapshot` for TiDB.

After everything is done, you can see the exported file in `/tmp/test`:

```
ls -lh /tmp/test | awk '{print $5 "\t" $9}'
```

```
140B metadata
66B test-schema-create.sql
300B test.sbttest1-schema.sql
190K test.sbttest1.0.sql
300B test.sbttest2-schema.sql
190K test.sbttest2.0.sql
300B test.sbttest3-schema.sql
190K test.sbttest3.0.sql
```

### 10.11.3.9 Export historical data snapshot of TiDB

Dumpling can export the data of a certain `tidb_snapshot` with the `--snapshot` option specified.

The `--snapshot` option can be set to a TSO (the `Position` field output by the `SHOW → MASTER STATUS` command) or a valid time of the `datetime` data type (in the form of `YYYY-MM-DD hh:mm:ss`), for example:

```
./dumpling --snapshot 417773951312461825
./dumpling --snapshot "2020-07-02 17:12:45"
```

The TiDB historical data snapshots when the TSO is 417773951312461825 and the time is 2020-07-02 17:12:45 are exported.

### 10.11.3.10 Control the memory usage of exporting large tables

When Dumpling is exporting a large single table from TiDB, Out of Memory (OOM) might occur because the exported data size is too large, which causes connection abort and export failure. You can use the following parameters to reduce the memory usage of TiDB:

- Setting `-r` to split the data to be exported into chunks. This reduces the memory overhead of TiDB's data scan and enables concurrent table data dump to improve export efficiency.
- Reduce the value of `--tidb-mem-quota-query` to 8589934592 (8 GB) or lower. `--→ tidb-mem-quota-query` controls the memory usage of a single query statement in TiDB.
- Adjust the `--params "tidb_distsql_scan_concurrency=5"` parameter. `tidb_distsql_scan_concurrency` is a session variable which controls the concurrency of the scan operations in TiDB.

### 10.11.3.11 TiDB GC settings when exporting a large volume of data

When exporting data from TiDB, if the TiDB version is later than or equal to v4.0.0 and Dumpling can access the PD address of the TiDB cluster, Dumpling automatically extends the GC time without affecting the original cluster.

In other scenarios, if the data size is very large, to avoid export failure due to GC during the export process, you can extend the GC time in advance:

```
SET GLOBAL tidb_gc_life_time = '720h';
```

After your operation is completed, set the GC time back (the default value is 10m):

```
SET GLOBAL tidb_gc_life_time = '10m';
```

Finally, all the exported data can be imported back to TiDB using [TiDB Lightning](#).

#### 10.11.4 Option list of Dumpling

---

Options Usage	Default value
-V or --version	Output the Dumpling version and exit directly
--	
↳ <b>version</b>	
↳	
-B or --database	Export specified databases
--	
↳ <b>database</b>	
↳	
-T or --tables	Export specified tables
--	
↳ <b>tables</b>	
↳ -	
↳ <b>list</b>	
↳	
-f or --filter	Export tables that match the filter pattern. For the filter syntax, see <a href="#">table-filter</a> .
↳ <b>filter</b>	
↳	
--	[\\*.\\*, !/^(\n        ↳ mysql\n        ↳ \';\n        ↳ sys\n        ↳ \';\n        ↳ INFORMATION_SCHEMA\n        ↳ \';\n        ↳ PERFORMANCE_SCHEMA\n        ↳ \';\n        ↳ METRICS_SCHEMA\n        ↳ \';\n        ↳ INSPECTION_SCHEMA\n        ↳ )\$\n        ↳ /.\\*)]\n        ↳\n        (export\n          all\n          databases\n          or tables\n          excluding\n          system\n          schemas)\n        false\n        (case-\n          insensitive)
--	whether table-filter is case-sensitive
↳ <b>case</b>	
↳ -	
↳ <b>sensitive</b>	
↳	

---

Options Usage	Default value
<b>-h</b> or <b>--host</b>	The IP address of the connected database host “127.0.0.1”
<b>--</b>	
<b>↳ host</b>	
<b>--</b>	
<b>-t</b> or <b>--threads</b>	The number of concurrent backup threads 4
<b>--</b>	
<b>↳ threads</b>	
<b>--</b>	
<b>-r</b> or <b>--rows</b>	Divide the table into specified rows of data (generally applicable for concurrent operations of splitting a large table into multiple files.)
<b>--</b>	
<b>↳ rows</b>	
<b>--</b>	
<b>-L</b> or <b>--logfile</b>	Log output address. If it is empty, the log will be output to the console “”
<b>--</b>	
<b>↳ logfile</b>	
<b>--</b>	
<b>-l</b> or <b>--loglevel</b>	Log level {debug,info,warn,error,dpanic,panic,fatal} “info”
<b>--</b>	
<b>↳ loglevel</b>	
<b>--</b>	
<b>-f</b> or <b>--logfmt</b>	Log output format {text,json} “text”
<b>--</b>	
<b>↳ logfmt</b>	
<b>--</b>	
<b>-d</b> or <b>--no-data</b>	Do not export data (suitable for scenarios where only the schema is exported)
<b>--</b>	
<b>↳ data</b>	
<b>--</b>	
<b>--no-header</b>	Export CSV files of the tables without generating header
<b>↳ header</b>	
<b>--</b>	
<b>-W</b> or <b>--no-views</b>	Do not export the views true
<b>--</b>	
<b>↳ views</b>	
<b>--</b>	
<b>-m</b> or <b>--no-schemas</b>	Do not export the schema with only the data exported
<b>--</b>	
<b>↳ schemas</b>	
<b>--</b>	

Options Usage	Default value
<b>-s</b> or <b>--statement</b>	Control the size of the <code>INSERT</code> statements; the unit is bytes
<b>--size</b>	
<b>-F</b> or <b>--filesize</b>	The file size of the divided tables. The unit must be specified such as 128B, 64KiB, 32MiB, and 1.5GiB.
<b>--filetype</b>	Exported file type (csv/sql)
<b>-o</b> or <b>--output</b>	The path of exported local files or the URL of the external storage
<b>-S</b> or <b>--sql</b>	Export data according to the specified SQL statement. This command does not support concurrent export.
<b>--consistency</b>	flush: use FTWRL before the dump snapshot: dump the TiDB data specific snapshot of a TSO lock: execute <code>lock tables read</code> on all tables to be dumped none: dump without adding locks, which cannot guarantee consistency auto: use <code>--consistency flush</code> for MySQL; use <code>--consistency snapshot</code> for TiDB
<b>--snapshot</b>	Snapshot TSO; valid only when <code>consistency=snapshot</code>
<b>--where</b>	Specify the scope of the table backup through the <code>where</code> condition
<b>-p</b> or <b>--password</b>	The password of the connected database host
<b>-P</b> or <b>--port</b>	The port of the connected database host
<b>-u</b> or <b>--user</b>	The username of the connected database host

---

Options Usage	Default value
-- Export the CREATE DATABASE statements of the empty databases → <b>dump</b> → - → <b>empty</b> → - → <b>database</b> →	true
--ca The address of the certificate authority file for TLS connection -- The address of the client certificate file for TLS connection → <b>cert</b> →	
--key The address of the client private key file for TLS connection --csv Delimiter of character type variables in CSV files → - → <b>delimiter</b> →	”,
--csv Separator of each value in CSV files. It is not recommended to use → - the default ‘,’. It is recommended to use ‘ + ’ or other uncommon → <b>separat<del>char</del>acter</b> combinations →	‘,’
--csv Representation of null values in CSV files → - → <b>null</b> → - → <b>value</b> →	“\N”
-- Use backslash (\) to escape special characters in the export file → <b>escape</b> → - → <b>backslash</b> →	true
-- The filename templates represented in the format of <a href="#">golang template</a> → <b>output</b> Support the {{.DB}}, {{.Table}}, and {{.Index}} arguments The → - three arguments represent the database name, table name, and → <b>filename</b> ID of the data file → - → <b>template</b> →	‘{{.DB}}.{{.Table}}.{{.Index}}’

Options Usage	Default value
-- Dumpling's service address, including the address for Prometheus to ":{port}"	"{port}"
→ <b>stat</b> ps all metrics and pprof debugging	
→ -	
→ <b>addr</b>	
→	
-- The memory limit of exporting SQL statements by a single line of 34359738368	34359738368
→ <b>tidb</b> Dumpling command, and the unit is byte. For v4.0.10 or later	
→ - versions, if you do not set this parameter, TiDB uses the value of	
→ <b>mem</b> the <b>mem-quota-query</b> configuration item as the memory limit value	
→ - by default. For versions earlier than v4.0.10, the parameter value	
→ <b>quota</b> defaults to 32 GB.	
→ -	
→ <b>query</b>	
→	
-- Specifies the session variable for the connection of the database to	
→ <b>params</b> exported. The required format is "character_set_client=	
→ → latin1,character_set_connection=latin1"	

## 10.12 sync-diff-inspector

### 10.12.1 sync-diff-inspector User Guide

[sync-diff-inspector](#) is a tool used to compare data stored in the databases with the MySQL protocol. For example, it can compare the data in MySQL with that in TiDB, the data in MySQL with that in MySQL, or the data in TiDB with that in TiDB. In addition, you can also use this tool to repair data in the scenario where a small amount of data is inconsistent.

This guide introduces the key features of sync-diff-inspector and describes how to configure and use this tool. To download sync-diff-inspector, use one of the following methods:

- Binary package. Click [tidb-community-toolkit-v5.3.0-linux-amd64](#) to download.
- Docker image. Execute the following command to download:

```
docker pull pingcap/tidb-enterprise-tools
```

#### 10.12.1.1 Key features

- Compare the table schema and data
- Generate the SQL statements used to repair data if the data inconsistency exists
- Support [data check for tables with different schema or table names](#)

- Support `data check` in the sharding scenario
- Support `data check` for TiDB upstream-downstream clusters
- Support `data check` in the DM replication scenario

#### 10.12.1.2 Restrictions of sync-diff-inspector

- Online check is not supported for data migration between MySQL and TiDB. Ensure that no data is written into the upstream-downstream checklist, and that data in a certain range is not changed. You can check data in this range by setting `range`.
- `JSON`, `BIT`, `BINARY`, `BLOB` and other types of data are not supported. When you perform a data check, you need to set `ignore-columns` to skip checking these types of data.
- In TiDB and MySQL, `FLOAT`, `DOUBLE` and other floating-point types are implemented differently. `FLOAT` and `DOUBLE` respectively take 6 and 15 significant digits for calculating checksum. If you do not want to use this feature, set `ignore-columns` to skip checking these columns.
- Support checking tables that do not contain the primary key or the unique index. However, if data is inconsistent, the generated SQL statements might not be able to repair the data correctly.

#### 10.12.1.3 Database privileges for sync-diff-inspector

sync-diff-inspector needs to obtain the information of table schema and to query data. The required database privileges are as follows:

- Upstream database
  - `SELECT` (checks data for comparison)
  - `SHOW DATABASES` (views database name)
  - `RELOAD` (views table schema)
- Downstream database
  - `SELECT` (checks data for comparison)
  - `SHOW DATABASES` (views database name)
  - `RELOAD` (views table schema)

#### 10.12.1.4 Configuration file description

The configuration of sync-diff-inspector consists of the following parts:

- **Global config:** General configurations, such as number of threads to check, whether to export SQL statement to fix inconsistent tables, and whether to compare the data.
- **Databases config:** Configures the instances of the upstream and downstream databases.

- **Routes:** Rules for upstream multiple schema names to match downstream single schema names (**optional**).
- **Task config:** Configures the tables for checking. If some tables have a certain mapping relationship between the upstream and downstream databases or have some special requirements, you must configure these tables.
- **Table config:** Special configurations for specific tables, such as specified ranges and columns to be ignored (**optional**).

Below is the description of a complete configuration file:

- Note: configurations with **s** after their name can have multiple values, so you need to use square brackets [] to contain the configuration values.

```
### Diff Configuration.

##### Global config #####
### The number of goroutines created to check data. The number of
    ↪ connections between sync-diff-inspector and upstream/downstream
    ↪ databases is slightly greater than this value.
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables.
export-fix-sql = true

### Only compares the table structure instead of the data.
check-struct-only = false

##### Datasource config #####
[data-sources]
[data-sources.mysql1] # mysql1 is the only custom ID for the database
    ↪ instance. It is used for the following `task.source-instances/task.
    ↪ target-instance` configuration.
    host = "127.0.0.1"
    port = 3306
    user = "root"
    password = ""

    # (optional) Use mapping rules to match multiple upstream sharded tables
        ↪ . Rule1 and rule2 are configured in the following Routes section.
    route-rules = ["rule1", "rule2"]

[data-sources.tidb0]
    host = "127.0.0.1"
    port = 4000
```

```

user = "root"
password = ""
# (optional) Uses the snapshot feature. If enabled, historical data is
    ↪ used for comparison
# snapshot = "386902609362944000"

##### Routes #####
### To compare the data of a large number of tables with different schema
    ↪ names or table names, or check the data of multiple upstream sharded
    ↪ tables and downstream table family, use the table-rule to configure
    ↪ the mapping relationship. You can configure the mapping rule only for
    ↪ the schema or table. Also, you can configure the mapping rules for
    ↪ both the schema and the table.

[routes]
[routes.rule1] # rule1 is the only custom ID for the configuration. It is
    ↪ used for the above `data-sources.route-rules` configuration.
schema-pattern = "test_*" # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "t_*"      # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test"     # The name of the schema in the target database
target-table = "t"          # The name of the target table
[routes.rule2]
schema-pattern = "test2_*" # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "t2_*"     # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test2"     # The name of the schema in the target database
target-table = "t2"          # The name of the target table

##### task config #####
### Configures the tables of the target database that need to be compared.

[task]
    # output-dir saves the following information:
    # 1 sql: The SQL file to fix tables that is generated after error is
        ↪ detected. One chunk corresponds to one SQL file.
    # 2 log: sync-diff.log
    # 3 summary: summary.txt
    # 4 checkpoint: a dir
    output-dir = "./output"
    # The upstream database. The value is the unique ID declared by data-
        ↪ sources.
source-instances = ["mysql1"]
    # The downstream database. The value is the unique ID declared by data-
        ↪ sources.

```

```

target-instance = "tidb0"
# The tables of downstream databases to be compared. Each table needs to
    ↪ contain the schema name and the table name, separated by '.'
# Use "?" to match any character and "*" to match characters of any
    ↪ length.
# For detailed match rules, refer to golang regexp pkg: https://github.
    ↪ com/google/re2/wiki/Syntax.
target-check-tables = ["schema*.table*", "!c.*", "test2.t2"]
# (optional) Extra configurations for some tables, Config1 is defined in
    ↪ the following table config example.
target-configs = ["config1"]

#####
##### Table config #####
#####

### Special configurations for specific tables. The tables to be configured
    ↪ must be in `task.target-check-tables`.
[table-configs.config1] # config1 is the only custom ID for this
    ↪ configuration. It is used for the above `task.target-configs`
    ↪ configuration.
### The name of the target table, you can use regular expressions to match
    ↪ multiple tables, but one table is not allowed to be matched by
    ↪ multiple special configurations at the same time.
target-tables = ["schema*.test*", "test2.t2"]
### (optional) Specifies the range of the data to be checked
### It needs to comply with the syntax of the WHERE clause in SQL.
range = "age > 10 AND age < 20"
### (optional) Specifies the column used to divide data into chunks. If you
    ↪ do not configure it,
### sync-diff-inspector chooses an appropriate column (primary key, unique
    ↪ key, or a field with index).
index-fields = ["col1","col2"]
### (optional) Ignores checking some columns such as some types (json, bit,
    ↪ blob, etc.)
### that sync-diff-inspector does not currently support.
### The floating-point data type behaves differently in TiDB and MySQL. You
    ↪ can use
### `ignore-columns` to skip checking these columns.
ignore-columns = [","",]
### (optional) Specifies the size of the chunk for dividing the table. If
    ↪ not specified, this configuration can be deleted or be set as 0.
chunk-size = 0
### (optional) Specifies the "collation" for the table. If not specified,
    ↪ this configuration can be deleted or be set as an empty string.
collation = ""

```

### 10.12.1.5 Run sync-diff-inspector

Run the following command:

```
./bin/sync_diff_inspector --config=./config.toml
```

This command outputs a check report `summary.txt` in the `output-dir` of `config.toml` and the log `sync_diff.log`. In the `output-dir`, a folder named by the hash value of the `config.toml` file is also generated. This folder includes the checkpoint node information of breakpoints and the SQL file generated when the data is inconsistent.

#### 10.12.1.5.1 Progress information

`sync-diff-inspector` sends progress information to `stdout` when running. Progress information includes the comparison results of table structures, comparison results of table data and the progress bar.

##### Note:

To ensure the display effect, keep the display window width above 80 characters.

A total of 2 tables need to be compared

```
Comparing the table structure of ``sbtest``.`sbtest96`` ... equivalent
Comparing the table structure of ``sbtest``.`sbtest99`` ... equivalent
Comparing the table data of ``sbtest``.`sbtest96`` ... failure
Comparing the table data of ``sbtest``.`sbtest99`` ...

-----
→
Progress [=====>---] 98%
→ 193/200
```

A total of 2 tables need to be compared

```
Comparing the table structure of ``sbtest``.`sbtest96`` ... equivalent
Comparing the table structure of ``sbtest``.`sbtest99`` ... equivalent
Comparing the table data of ``sbtest``.`sbtest96`` ... failure
Comparing the table data of ``sbtest``.`sbtest99`` ... failure

-----
```

```
→
Progress [=====>]
→ 100% 0/0
```

```
The data of `sbtest`.`sbtest99` is not equal
The data of `sbtest`.`sbtest96` is not equal

The rest of tables are all equal.
The patch file has been generated in
    'output/fix-on-tidb2/'
You can view the comparision details through 'output/sync_diff.log'
```

#### 10.12.1.5.2 Output file

The directory structure of the output file is as follows:

```
output/
|-- checkpoint # Saves the breakpoint information
| |-- bbfec8cc8d1f58a5800e63aa73e5 # Config hash. The placeholder file which
|   ↪ identifies the configuration file corresponding to the output
|   ↪ directory (output/)
|-- DO_NOT_EDIT_THIS_DIR
-- sync_diff_checkpoints.pb # The breakpoint information
|
|-- fix-on-target # Saves SQL files to fix data inconsistency
| |-- xxx.sql
| |-- xxx.sql
| -- xxx.sql
|
|-- summary.txt # Saves the summary of the check results
-- sync_diff.log # Saves the output log informatiion when sync-diff-
  ↪ inspector is running
```

#### 10.12.1.5.3 Log

The log of sync-diff-inspector is saved in \${output}/sync\_diff.log, among which \${output} is the value of output-dir in the config.toml file.

#### 10.12.1.5.4 Progress

The running sync-diff-inspector periodically (every 10 seconds) prints the progress in checkpoint, which is located at \${output}/checkpoint/sync\_diff\_checkpoints.pb, among which \${output} is the value of output-dir in the config.toml file.

#### 10.12.1.5.5 Result

After the check is finished, sync-diff-inspector outputs a report. It is located at \${output}/summary.txt, and \${output} is the value of output-dir in the config.toml file.

TABLE	STRUCTURE EQUALITY	DATA DIFF ROWS
`sbtest`.`sbtest99`	true	+97/-97
`sbtest`.`sbtest96`	true	+0/-101

Time Cost: 16.75370462s  
Average Speed: 113.277149MB/s

- TABLE: The corresponding database and table names
- STRUCTURE EQUALITY: Checks whether the table structure is the same
- DATA DIFF ROWS: `rowAdd` / `rowDelete`. Indicates the number of rows that need to be added/deleted to fix the table

#### 10.12.1.5.6 SQL statements to fix inconsistent data

If different rows exist during the data checking process, the SQL statements will be generated to fix them. If the data inconsistency exists in a chunk, a SQL file named by `chunk.Index` will be generated. The SQL file is located at  `${output}/fix-on-${instance}` , and  `${instance}`  is the value of `task.target-instance` in the `config.toml` file.

A SQL file contains the tale to which the chunk belong and the range information. For the SQL files, you should consider the following three situations:

- If the rows in the downstream database are missing, REPLACE statements will be applied
- If the rows in the downstream database are redundant, DELETE statements will be applied
- If some data of the rows in the downstream database is inconsistent, REPLACE statements will be applied and inconsistent columns will be marked with annotation in the SQL file

```
-- table: sbtest.sbtest99
-- range in sequence: (3690708) < (id) <= (3720581)
/*
DIFF COLUMNS  `K`          `C`          `PAD` 
→
source data  2501808  'hello'           'world'
→
target data  5003616  '0709824117-9809973320-4456050422'
→ '1714066100-7057807621-1425865505'
```

```

    ↵
*/  

REPLACE INTO `sbtest`.`sbtest99`(`id`, `k`, `c`, `pad`) VALUES  

    ↵ (3700000, 2501808, 'hello', 'world');

```

#### 10.12.1.6 Note

- sync-diff-inspector consumes a certain amount of server resources when checking data. Avoid using sync-diff-inspector to check data during peak business hours.
- TiDB uses the `utf8_bin` collation. If you need to compare the data in MySQL with that in TiDB, pay attention to the collation configuration of MySQL tables. If the primary key or unique key is the `varchar` type and the collation configuration in MySQL differs from that in TiDB, then the final check result might be incorrect because of the collation issue. You need to add collation to the sync-diff-inspector configuration file.
- sync-diff-inspector divides data into chunks first according to TiDB statistics and you need to guarantee the accuracy of the statistics. You can manually run the `analyze ↵ table {table_name}` command when the TiDB server's *workload is light*.
- Pay special attention to `table-rules`. If you configure `schema-pattern="test1"`, `table-pattern = "t_1"`, `target-schema="test2"` and `target-table = "t_2"`, the `test1.t_1` schema in the source database and the `test2.t_2` schema in the target database are compared. Sharding is enabled by default in sync-diff-inspector, so if the source database has a `test2.t_2` table, the `test1.t_1` table and `test2.t_2` table in the source database serving as sharding are compared with the `test2.t_2` table in the target database.
- The generated SQL file is only used as a reference for repairing data, and you need to confirm it before executing these SQL statements to repair data.

#### 10.12.2 Data Check for Tables with Different Schema or Table Names

When using replication tools such as [TiDB Data Migration](#), you can set `route-rules` to replicate data to a specified table in the downstream. sync-diff-inspector enables you to verify tables with different schema names or table names by setting `rules`.

The following is a simple configuration example. To learn the complete configuration, refer to [Sync-diff-inspector User Guide](#).

```
#####
# Datasource config #####
[datasources.mysql1]
host = "127.0.0.1"
port = 3306
user = "root"
password = ""
```

```

route-rules = ["rule1"]

[data-sources.tidb0]
host = "127.0.0.1"
port = 4000
user = "root"
password = ""
##### Routes #####
[routes.rule1]
schema-pattern = "test_1" # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "t_1"      # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test_2"    # The name of the schema in the target database
target-table = "t_2"        # The name of the target table

```

This configuration can be used to check `test_2.t_2` in the downstream and `test_1.t_1` in the `mysql1` instance.

To check a large number of tables with different schema names or table names, you can simplify the configuration by setting the mapping relationship by using `rules`. You can configure the mapping relationship of either schema or table, or of both. For example, all the tables in the upstream `test_1` database are replicated to the downstream `test_2` database, which can be checked through the following configuration:

```

##### Datasource config #####
[data-sources.mysql1]
host = "127.0.0.1"
port = 3306
user = "root"
password = ""
route-rules = ["rule1"]

[data-sources.tidb0]
host = "127.0.0.1"
port = 4000
user = "root"
password = ""
##### Routes #####
[routes.rule1]
schema-pattern = "test_1" # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "*"       # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test_2"    # The name of the schema in the target database
target-table = "t_2"        # The name of the target table

```

### 10.12.2.1 Note

If `test_2.t_2` exists in the upstream database, the downstream database also compares this table.

### 10.12.3 Data Check in the Sharding Scenario

sync-diff-inspector supports data check in the sharding scenario. Assume that you use the [TiDB Data Migration](#) tool to replicate data from multiple MySQL instances into TiDB, you can use sync-diff-inspector to check upstream and downstream data.

For scenarios where the number of upstream sharded tables is small and the naming rules of sharded tables do not have a pattern as shown below, you can use `Datasource config` to configure `table-0`, set corresponding `rules` and configure the tables that have the mapping relationship between the upstream and downstream databases. This configuration method requires setting all sharded tables.

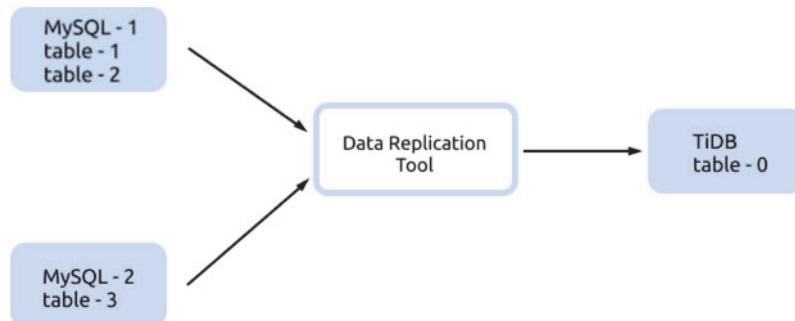


Figure 166: shard-table-replica-1

Below is a complete example of the sync-diff-inspector configuration.

```

### Diff Configuration.

#####
# Global config #####
#####

### The number of goroutines created to check data. The number of
# connections between upstream and downstream databases are slightly
# greater than this value
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables
export-fix-sql = true

### Only compares the table structure instead of the data
check-struct-only = false
  
```

```
#####
# Datasource config #####
[datasources.mysql1]
  host = "127.0.0.1"
  port = 3306
  user = "root"
  password = ""

  route-rules = ["rule1"]

[datasources.mysql2]
  host = "127.0.0.1"
  port = 3306
  user = "root"
  password = ""

  route-rules = ["rule2"]

[datasources.tidb0]
  host = "127.0.0.1"
  port = 4000
  user = "root"
  password = ""

#####
# Routes #####
[routes.rule1]
schema-pattern = "test"      # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "table-[1-2]" # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test"        # The name of the schema in the target database
target-table = "table-0"       # The name of the target table

[routes.rule2]
schema-pattern = "test"      # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "table-3"     # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test"        # The name of the schema in the target database
target-table = "table-0"       # The name of the target table

#####
# Task config #####
[task]
  output-dir = "./output"
```

```

source-instances = ["mysql11", "mysql12"]

target-instance = ["tidb0"]

# The tables of downstream databases to be compared. Each table needs to
# contain the schema name and the table name, separated by '.'
target-check-tables = ["test.table-0"]

```

You can use `table-rules` for configuration when there are a large number of upstream sharded tables and the naming rules of all sharded tables have a pattern, as shown below:

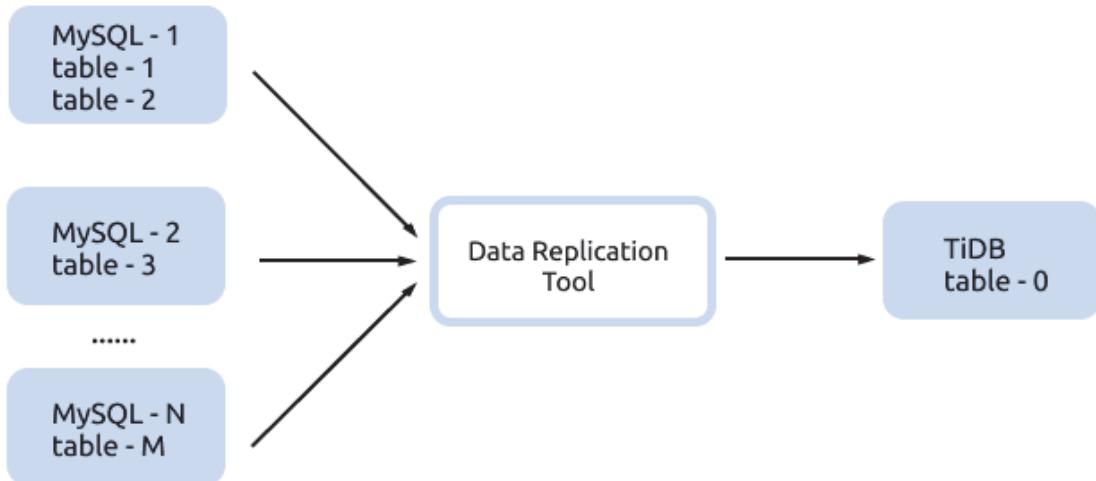


Figure 167: shard-table-replica-2

Below is a complete example of the sync-diff-inspector configuration.

```

### Diff Configuration.

#####
##### Global config #####
#####

### The number of goroutines created to check data. The number of
# connections between upstream and downstream databases are slightly
# greater than this value.
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables.
export-fix-sql = true

### Only compares the table structure instead of the data.
check-struct-only = false

```

```
#####
# Datasource config #####
[datasources.mysql1]
host = "127.0.0.1"
port = 3306
user = "root"
password = ""

[datasources.mysql12]
host = "127.0.0.1"
port = 3306
user = "root"
password = ""

[datasources.tidb0]
host = "127.0.0.1"
port = 4000
user = "root"
password = ""

#####
# Routes #####
[routes.rule1]
schema-pattern = "test" # Matches the schema name of the data source.
    ↪ Supports the wildcards "*" and "?"
table-pattern = "table-*" # Matches the table name of the data source.
    ↪ Supports the wildcards "*" and "?"
target-schema = "test" # The name of the schema in the target database
target-table = "table-0" # The name of the target table

#####
# Task config #####
[task]
output-dir = "./output"
source-instances = ["mysql1", "mysql12"]

target-instance = ["tidb0"]

# The tables of downstream databases to be compared. Each table needs to
    ↪ contain the schema name and the table name, separated by '.'
target-check-tables = ["test.table-0"]
```

### 10.12.3.1 Note

If `test.table-0` exists in the upstream database, the downstream database also compares this table.

#### 10.12.4 Data Check for TiDB Upstream and Downstream Clusters

You can use TiDB Binlog to build upstream and downstream clusters of TiDB. When Drainer replicates data to TiDB, the checkpoint is saved and the TSO mapping relationship between the upstream and downstream is also saved as `ts-map`. To check data between the upstream and downstream, configure `snapshot` in sync-diff-inspector.

##### 10.12.4.1 Step 1: obtain `ts-map`

To obtain `ts-map`, execute the following SQL statement in the downstream TiDB cluster:

```
mysql> select * from tidb_binlog.checkpoint;
+-----+-----+
| clusterID | checkPoint |
+-----+-----+
| 6711243465327639221 | {"commitTS":409622383615541249,"ts-map":{"primary-ts":409621863377928194,"secondary-ts":409621863377928345}} |
+-----+-----+
```

##### 10.12.4.2 Step 2: configure snapshot

Then configure the snapshot information of the upstream and downstream databases by using the `ts-map` information obtained in [Step 1](#).

Here is a configuration example of the `Datasource config` section:

```
##### Datasource config #####
[data-sources.mysql1]
host = "127.0.0.1"
port = 3306
user = "root"
password = ""
snapshot = "409621863377928345"
[data-sources.tidb0]
host = "127.0.0.1"
port = 4000
user = "root"
snapshot = "409621863377928345"
```

**Note:**

- Set `db-type` of Drainer to `tidb` to ensure that `ts-map` is saved in the checkpoint.
- Modify the Garbage Collection (GC) time of TiKV to ensure that the historical data corresponding to snapshot is not collected by GC during the data check. It is recommended that you modify the GC time to 1 hour and recover the setting after the check.
- In some versions of TiDB Binlog, `master-ts` and `slave-ts` are stored in `ts-map`. `master-ts` is equivalent to `primary-ts` and `slave-ts` is equivalent to `secondary-ts`.
- The above example only shows the section of `Datasource config`. For complete configuration, refer to [sync-diff-inspector User Guide](#).

#### 10.12.5 Data Check in the DM Replication Scenario

When using replication tools such as [TiDB Data Migration](#), you need to check the data consistency before and after the replication process. You can set a specific `task-name` configuration from `DM-master` to perform a data check.

The following is a simple configuration example. To learn the complete configuration, refer to [Sync-diff-inspector User Guide](#).

```
### Diff Configuration.

#####
# Global config #####
#####

### The number of goroutines created to check data. The number of
#→ connections between upstream and downstream databases are slightly
#→ greater than this value.
check-thread-count = 4

### If enabled, SQL statements is exported to fix inconsistent tables.
export-fix-sql = true

### Only compares the table structure instead of the data.
check-struct-only = false

### The IP address of dm-master and the format is "http://127.0.0.1:8261".
dm-addr = "http://127.0.0.1:8261"

### Specifies the `task-name` of DM.
```

```

dm-task = "test"

#####
# Task config #####
[task]
    output-dir = "./output"

    # The tables of downstream databases to be compared. Each table needs to
    # contain the schema name and the table name, separated by '.'
    target-check-tables = ["hb_test.*"]

```

This example is configured in dm-task = “test”, which checks all the tables of hb\_test schema under the “test” task. It automatically gets the regular matching of the schemas between upstream and downstream databases to verify the data consistency after DM replication.

## 10.13 TiSpark

### 10.13.1 TiSpark Quick Start Guide

To make it easy to [try TiSpark](#), the TiDB cluster installed using TiUP integrates Spark and TiSpark jar package by default.

#### 10.13.1.1 Deployment information

- Spark is deployed by default in the `spark` folder in the TiDB instance deployment directory.
- The TiSpark jar package is deployed by default in the `jars` folder in the Spark deployment directory.

```
spark/jars/tispark-${name_with_version}.jar
```

- TiSpark sample data and import scripts can be downloaded from [TiSpark sample data](#).

```
tispark-sample-data/
```

#### 10.13.1.2 Prepare the environment

##### 10.13.1.2.1 Install JDK on the TiDB instance

Download the latest version of JDK 1.8 from [Oracle JDK official download page](#). The version used in the following example is `jdk-8u141-linux-x64.tar.gz`.

Extract the package and set the environment variables based on your JDK deployment directory.

Edit the `~/.bashrc` file. For example:

```
export JAVA_HOME=/home/pingcap/jdk1.8.0_144
export PATH=$JAVA_HOME/bin:$PATH
```

Verify the validity of JDK:

```
$ java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
```

#### 10.13.1.2.2 Import the sample data

Assume that the TiDB cluster is started. The service IP of one TiDB instance is 192.168.0.2, the port is 4000, the user name is `root`, and the password is null.

```
wget http://download.pingcap.org/tispark-sample-data.tar.gz
tar -zvxf tispark-sample-data.tar.gz
cd tispark-sample-data
```

Edit the TiDB login information in `sample_data.sh`. For example:

```
mysql --local-infile=1 -h 192.168.0.2 -P 4000 -u root < dss.ddl
```

Run the script:

```
./sample_data.sh
```

#### Note:

You need to install the MySQL client on the machine that runs the script. If you are a CentOS user, you can install it through the command `yum -y → install mysql`.

Log into TiDB and verify that the `TPCH_001` database and the following tables are included.

```
$ mysql -uroot -P4000 -h192.168.0.2
MySQL [(none)]> show databases;
+-----+
| Database      |
+-----+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
```

```

| TPCH_001      |
| mysql         |
| test          |
+-----+
5 rows in set (0.00 sec)

MySQL [(none)]> use TPCH_001
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [TPCH_001]> show tables;
+-----+
| Tables_in_TPCH_001 |
+-----+
| CUSTOMER           |
| LINEITEM           |
| NATION             |
| ORDERS             |
| PART               |
| PARTSUPP           |
| REGION             |
| SUPPLIER           |
+-----+
8 rows in set (0.00 sec)

```

#### 10.13.1.3 Use example

First start the spark-shell:

```
$ cd spark
$ bin/spark-shell
```

Then query the TiDB table as you are using the native Spark SQL:

```
scala> spark.sql("use TPCH_001")
scala> spark.sql("select count(*) from lineitem").show
```

The result is:

```
+-----+
| count(1) |
+-----+
|   60175 |
+-----+
```

Now run a more complex Spark SQL:

```
scala> spark.sql(
    """select
       |   l_returnflag,
       |   l_linenstatus,
       |   sum(l_quantity) as sum_qty,
       |   sum(l_extendedprice) as sum_base_price,
       |   sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
       |   sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as
       |     sum_charge,
       |   avg(l_quantity) as avg_qty,
       |   avg(l_extendedprice) as avg_price,
       |   avg(l_discount) as avg_disc,
       |   count(*) as count_order
      |from
      |  lineitem
      |where
      |  l_shipdate <= date '1998-12-01' - interval '90' day
      |group by
      |  l_returnflag,
      |  l_linenstatus
      |order by
      |  l_returnflag,
      |  l_linenstatus
    """.stripMargin).show
```

The result is:

l_returnflag	l_linenstatus	sum_qty	sum_base_price	sum_disc_price
A	F	380456.00	532348211.65	505822441.4861
N	F	8971.00	12384801.37	11798257.2080
N	O	742802.00	1041502841.45	989737518.6346
R	F	381449.00	534594445.35	507996454.4067

(Continued)

sum_charge	avg_qty	avg_price	avg_disc	count_order
526165934.000839	25.575155	35785.709307	0.050081	14876
12282485.056933	25.778736	35588.509684	0.047759	348
1029418531.523350	25.454988	35691.129209	0.049931	29181
528524219.358903	25.597168	35874.006533	0.049828	14902

See [more examples](#).

### 10.13.2 TiSpark User Guide

TiSpark is a thin layer built for running Apache Spark on top of TiDB/TiKV to answer the complex OLAP queries. It takes advantages of both the Spark platform and the distributed TiKV cluster and seamlessly glues to TiDB, the distributed OLTP database, to provide a Hybrid Transactional/Analytical Processing (HTAP) solution to serve as a one-stop solution for both online transactions and analysis.

TiSpark depends on the TiKV cluster and the PD cluster. You also need to set up a Spark cluster. This document provides a brief introduction to how to setup and use TiSpark. It requires some basic knowledge of Apache Spark. For more information, see [Spark website](#).

#### 10.13.2.1 Overview

TiSpark is an OLAP solution that runs Spark SQL directly on TiKV, the distributed storage engine.

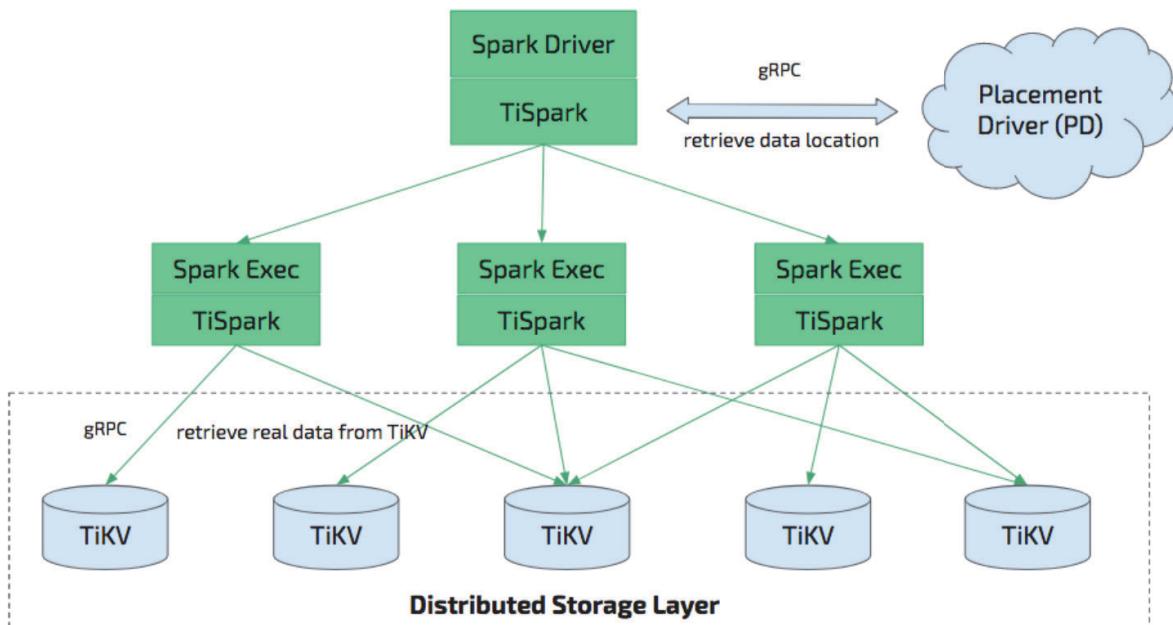


Figure 168: TiSpark architecture

- TiSpark integrates with Spark Catalyst Engine deeply. It provides precise control of the computing, which allows Spark read data from TiKV efficiently. It also supports index seek, which improves the performance of the point query execution significantly.

- It utilizes several strategies to push down the computing to reduce the size of dataset handling by Spark SQL, which accelerates the query execution. It also uses the TiDB built-in statistical information for the query plan optimization.
- From the data integration point of view, TiSpark and TiDB serve as a solution for running both transaction and analysis directly on the same platform without building and maintaining any ETLs. It simplifies the system architecture and reduces the cost of maintenance.
- You can deploy and utilize tools from the Spark ecosystem for further data processing and manipulation on TiDB. For example, using TiSpark for data analysis and ETL; retrieving data from TiKV as a machine learning data source; generating reports from the scheduling system and so on.
- Also, TiSpark supports distributed writes to TiKV. Compared to using Spark combined with JDBC to write to TiDB, distributed writes to TiKV can implement transactions (either all data are written successfully or all writes fail), and the writes are faster.

#### 10.13.2.2 Environment setup

- The TiSpark 2.x supports Spark 2.3.x and Spark 2.4.x. If you want to use Spark 2.1.x, use TiSpark 1.x instead.
- TiSpark requires JDK 1.8+ and Scala 2.11 (Spark2.0 + default Scala version).
- TiSpark runs in any Spark mode such as YARN, Mesos, and Standalone.

#### 10.13.2.3 Recommended configuration

This section describes the configuration of independent deployment of TiKV and TiSpark, independent deployment of Spark and TiSpark, and hybrid deployment of TiKV and TiSpark.

##### 10.13.2.3.1 Configuration of independent deployment of TiKV and TiSpark

For independent deployment of TiKV and TiSpark, it is recommended to refer to the following recommendations:

- Hardware configuration
  - For general purposes, refer to the TiDB and TiKV hardware configuration [recommendations](#).
  - If the usage is more focused on the analysis scenarios, you can increase the memory of the TiKV nodes to at least 64G.

##### 10.13.2.3.2 Configuration of independent deployment of Spark and TiSpark

See the [Spark official website](#) for the detail hardware recommendations.

The following is a short overview of TiSpark configuration.

It is recommended to allocate 32G memory for Spark, and reserve at least 25% of the memory for the operating system and buffer cache.

It is recommended to provision at least 8 to 16 cores on per machine for Spark. Initially, you can assign all the CPU cores to Spark.

See the [official configuration](#) on the Spark website. The following is an example based on the `spark-env.sh` configuration:

```
SPARK_EXECUTOR_CORES: 5  
SPARK_EXECUTOR_MEMORY: 10g  
SPARK_WORKER_CORES: 5  
SPARK_WORKER_MEMORY: 10g
```

In the `spark-defaults.conf` file, add the following lines:

```
spark.tispark.pd.addresses $your_pd_servers  
spark.sql.extensions org.apache.spark.sql.TiExtensions
```

Add the following configuration in the CDH spark version:

```
spark.tispark.pd.addresses=$your_pd_servers  
spark.sql.extensions=org.apache.spark.sql.TiExtensions
```

`your_pd_servers` are comma-separated PD addresses, with each in the format of `$your_pd_address:$port`.

For example, when you have multiple PD servers on 10.16.20.1, 10.16.20.2, 10.16.20.3 → with the port 2379, put it as 10.16.20.1:2379, 10.16.20.2:2379, 10.16.20.3:2379.

#### 10.13.2.3.3 Configuration of hybrid deployment of TiKV and TiSpark

For the hybrid deployment of TiKV and TiSpark, add TiSpark required resources to the TiKV reserved resources, and allocate 25% of the memory for the system.

#### 10.13.2.4 Deploy the TiSpark cluster

Download TiSpark's jar package [here](#). Download your desired version of jar package and copy the content to the appropriate folder.

##### 10.13.2.4.1 Deploy TiSpark on the existing Spark cluster

Running TiSpark on an existing Spark cluster does not require a reboot of the cluster. You can use Spark's `--jars` parameter to introduce TiSpark as a dependency:

```
spark-shell --jars $TISPARK_FOLDER/tispark-${name_with_version}.jar
```

#### 10.13.2.4.2 Deploy TiSpark without the Spark cluster

If you do not have a Spark cluster, we recommend using the standalone mode. To use the Spark Standalone model, you can simply place a compiled version of Spark on each node of the cluster. If you encounter problems, see its [official website](#). And you are welcome to [file an issue](#) on our GitHub.

Download and install

You can download [Apache Spark](#)

For the Standalone mode without Hadoop support, use Spark **2.3.x** and any version of Pre-build with Apache Hadoop 2.x with Hadoop dependencies. If you need to use the Hadoop cluster, choose the corresponding Hadoop version. You can also choose to build from the [source code](#) to match the previous version of the official Hadoop 2.x.

Suppose you already have a Spark binaries, and the current PATH is **SPARKPATH**, you can copy the TiSpark jar package to the  **\${SPARKPATH}/jars** directory.

Start a Master node

Execute the following command on the selected Spark Master node:

```
cd $SPARKPATH  
./sbin/start-master.sh
```

After the above step is completed, a log file will be printed on the screen. Check the log file to confirm whether the Spark-Master is started successfully. You can open the <http://spark-master-hostname:8080> to view the cluster information (if you does not change the Spark-Master default port number). When you start Spark-Worker, you can also use this panel to confirm whether the Worker is joined to the cluster.

Start a Worker node

Similarly, you can start a Spark-Worker node with the following command:

```
./sbin/start-slave.sh spark://spark-master-hostname:7077
```

After the command returns, you can see if the Worker node is joined to the Spark cluster correctly from the panel as well. Repeat the above command at all Worker nodes. After all Workers are connected to the master, you have a Standalone mode Spark cluster.

Spark SQL shell and JDBC server

TiSpark supports Spark 2.3, so you can use Spark's ThriftServer and SparkSQL directly.

#### 10.13.2.5 Demo

Assuming that you have successfully started the TiSpark cluster as described above, here's a quick introduction to how to use Spark SQL for OLAP analysis. Here we use a table named `lineitem` in the `tpch` database as an example.

Assuming that your PD node is located at 192.168.1.100, port 2379, add the following command to `$SPARK_HOME/conf/spark-defaults.conf`:

```
spark.tispark.pd.addresses 192.168.1.100:2379
spark.sql.extensions org.apache.spark.sql.TiExtensions
```

And then enter the following command in the Spark-Shell as in native Apache Spark:

```
spark.sql("use tpch")
spark.sql("select count(*)from lineitem").show
```

The result is:

```
+-----+
| Count (1) |
+-----+
| 6000000000 |
+-----+
```

Spark SQL Interactive shell remains the same:

```
spark-sql> use tpch;
Time taken: 0.015 seconds

spark-sql> select count(*) from lineitem;
2000
Time taken: 0.673 seconds, Fetched 1 row(s)
```

For JDBC connection with Thrift Server, you can try it with various JDBC supported tools including SQuirreLSQL and hive-beeline. For example, to use it with beeline:

```
./beeline
Beeline version 1.2.2 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000

1: jdbc:hive2://localhost:10000> use testdb;
+-----+
| Result |
+-----+
+-----+
No rows selected (0.013 seconds)

select count(*) from account;
+-----+
| count(1) |
+-----+
```

```
| 1000000 |
+-----+
1 row selected (1.97 seconds)
```

### 10.13.2.6 Use TiSpark together with Hive

You can use TiSpark together with Hive.

Before starting Spark, you need to set the `HADOOP_CONF_DIR` environment variable to your Hadoop configuration folder and copy `hive-site.xml` to the `spark/conf` folder.

```
val tisparkDF = spark.sql("select * from tispark_table").toDF
tisparkDF.write.saveAsTable("hive_table") // save table to hive
spark.sql("select * from hive_table a, tispark_table b where a.col1 = b.col1
    → ").show // join table across Hive and Tispark
```

### 10.13.2.7 Batch write DataFrames into TiDB using TiSpark

Starting from v2.3, TiSpark natively supports batch writing DataFrames into TiDB clusters. This writing mode is implemented through the two-phase commit protocol of TiKV.

Compared with the writing through Spark + JDBC, the TiSpark batch writing has the following advantages:

Aspects	TiSpark batch writes	Spark + JDBC writes
Atomicity	The DataFrames either are all written successfully or all fail to write.	If the Spark task fails and exits during the writing process, a part of the data might be written successfully.
Isolation	During the writing process, the data being written is invisible to other transactions.	During the writing process, some successfully written data is visible to other transactions.

Aspects to com- pare	TiSpark batch writes	Spark + JDBC writes
Error recovery	If the batch write fails, you only need to re-run Spark.	An application is required to achieve idempotence. For example, if the batch write fails, you need to clean up the part of the successfully written data and re-run Spark. You need to set <code>spark.task.maxFailures = 1</code> to prevent data duplication caused by task retry.
Speed	Data is directly written into TiKV, which is faster.	Data is written to TiKV through TiDB, which affects the speed.

The following example shows how to batch write data using TiSpark via the scala API:

```
// select data to write
val df = spark.sql("select * from tpch.ORDERS")

// write data to tidb
df.write.
  format("tidb").
  option("tidb.addr", "127.0.0.1").
  option("tidb.port", "4000").
  option("tidb.user", "root").
  option("tidb.password", "").
  option("database", "tpch").
  option("table", "target_orders").
```

```
mode("append").
save()
```

If the amount of data to write is large and the writing time exceeds ten minutes, you need to ensure that the GC time is longer than the writing time.

```
update mysql.tidb set VARIABLE_VALUE="6h" where VARIABLE_NAME=
↪ tikv_gc_life_time;
```

Refer to [this document](#) for details.

#### 10.13.2.8 Load Spark Dataframe into TiDB using JDBC

In addition to using TiSpark to batch write DataFrames into the TiDB cluster, you can also use Spark's native JDBC support for the data writing:

```
import org.apache.spark.sql.execution.datasources.jdbc.JDBCOptions

val customer = spark.sql("select * from customer limit 100000")
// You might repartition the source to make it balance across nodes
// and increase the concurrency.
val df = customer.repartition(32)
df.write
.mode(saveMode = "append")
.format("jdbc")
.option("driver", "com.mysql.jdbc.Driver")
// Replace the host and port with that of your own and be sure to use the
↪ rewrite batch
.option("url", "jdbc:mysql://127.0.0.1:4000/test?rewriteBatchedStatements=
↪ true")
.option("useSSL", "false")
// As tested, 150 is good practice
.option(JDBCOptions.JDBC_BATCH_INSERT_SIZE, 150)
.option("dbtable", s"cust_test_select") // database name and table name
↪ here
.option("isolationLevel", "NONE") // recommended to set isolationLevel to
↪ NONE if you have a large DF to load.
.option("user", "root") // TiDB user here
.save()
```

It is recommended to set `isolationLevel` to `NONE` to avoid large single transactions which might potentially lead to TiDB OOM.

**Note:**

When you use JDBC, the default value of `isolationLevel` is `READ_UNCOMMITTED`, which causes the error of unsupported isolation level transactions. It is recommended to set the value of `isolationLevel` to `NONE`.

### 10.13.2.9 Statistics information

TiSpark uses TiDB statistic information for the following items:

1. Determining which index to use in your query plan with the estimated lowest cost.
2. Small table broadcasting, which enables efficient broadcast join.

If you would like TiSpark to use statistic information, first you need to make sure that concerning tables have already been analyzed. Read more about [how to analyze tables](#).

Starting from TiSpark 2.0, statistics information is default to auto load.

Note that table statistics are cached in the memory of your Spark driver node, so you need to make sure that your memory size is large enough for your statistics information.

Currently, you can adjust these configurations in your `spark-defaults.conf` file.

---

Property	Default	Description
<code>spark.tispark.statsLoad</code>	<code>Whiteload</code>	to load statis- tics infor- ma- tion auto- mati- cally dur- ing database map- ping.

---

### 10.13.2.10 FAQ

Q: What are the pros/cons of independent deployment as opposed to a shared resource with an existing Spark / Hadoop cluster?

A: You can use the existing Spark cluster without a separate deployment, but if the existing cluster is busy, TiSpark will not be able to achieve the desired speed.

Q: Can I mix Spark with TiKV?

A: If TiDB and TiKV are overloaded and run critical online tasks, consider deploying TiSpark separately. You also need to consider using different NICs to ensure that OLTP's network resources are not compromised and affect online business. If the online business requirements are not high or the loading is not large enough, you can consider mixing TiSpark with TiKV deployment.

Q: What can I do if `WARN ObjectStore:568 - Failed to get database` is returned when executing SQL statements using TiSpark?

A: You can ignore this warning. It occurs because Spark tries to load two nonexistent databases (`default` and `global_temp`) in its catalog. If you want to mute this warning, modify `log4j` by adding `log4j.logger.org.apache.hadoop.hive.metastore.ObjectStore=ERROR` to the `log4j` file in `tispark/conf`. You can add the parameter to the `log4j` file of the `config` under Spark. If the suffix is `template`, you can use the `mv` command to change it to `properties`.

Q: What can I do if `java.sql.BatchUpdateException: Data Truncated` is returned when executing SQL statements using TiSpark?

A: This error occurs because the length of the data written exceeds the length of the data type defined by the database. You can check the field length and adjust it accordingly.

Q: Does TiSpark read Hive metadata by default?

A: By default, TiSpark searches for the Hive database by reading the Hive metadata in `hive-site`. If the search task fails, it searches for the TiDB database instead, by reading the TiDB metadata.

If you do not need this default behavior, do not configure the Hive metadata in `hive-site`.

Q: What can I do if `Error: java.io.InvalidClassException: com.pingcap.tikv.region.TiRegion; local class incompatible: stream classdesc serialVersionUID ...` is returned when TiSpark is executing a Spark task?

A: The error message shows a `serialVersionUID` conflict, which occurs because you have used `class` and `TiRegion` of different versions. Because `TiRegion` only exists in TiSpark, multiple versions of TiSpark packages might be used. To fix this error, you need to make sure the version of TiSpark dependency is consistent among all nodes in the cluster.

# 11 Reference

## 11.1 Cluster Architecture

### 11.1.1 TiDB Architecture

Compared with the traditional standalone databases, TiDB has the following advantages:

- Has a distributed architecture with flexible and elastic scalability.
- Fully compatible with the MySQL 5.7 protocol, common features and syntax of MySQL. To migrate your applications to TiDB, you do not need to change a single line of code in many cases.
- Supports high availability with automatic failover when a minority of replicas fail; transparent to applications.
- Supports ACID transactions, suitable for scenarios requiring strong consistency such as bank transfer.
- Provides a rich series of [data migration tools](#) for migrating, replicating, or backing up data.

As a distributed database, TiDB is designed to consist of multiple components. These components communicate with each other and form a complete TiDB system. The architecture is as follows:

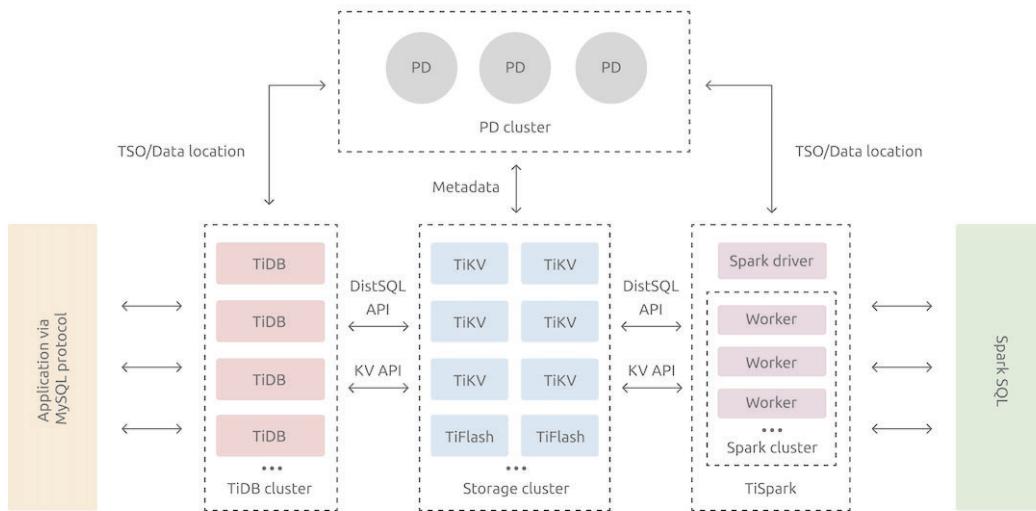


Figure 169: TiDB Architecture

#### 11.1.1.1 TiDB server

The TiDB server is a stateless SQL layer that exposes the connection endpoint of the MySQL protocol to the outside. The TiDB server receives SQL requests, performs SQL parsing and optimization, and ultimately generates a distributed execution plan. It is horizontally scalable and provides the unified interface to the outside through the load balancing components such as Linux Virtual Server (LVS), HAProxy, or F5. It does not store data and is only for computing and SQL analyzing, transmitting actual data read request to TiKV nodes (or TiFlash nodes).

#### 11.1.1.2 Placement Driver (PD) server

The PD server is the metadata managing component of the entire cluster. It stores metadata of real-time data distribution of every single TiKV node and the topology structure of the entire TiDB cluster, provides the TiDB Dashboard management UI, and allocates transaction IDs to distributed transactions. The PD server is “the brain” of the entire TiDB cluster because it not only stores metadata of the cluster, but also sends data scheduling command to specific TiKV nodes according to the data distribution state reported by TiKV nodes in real time. In addition, the PD server consists of three nodes at least and has high availability. It is recommended to deploy an odd number of PD nodes.

#### 11.1.1.3 Storage servers

##### 11.1.1.3.1 TiKV server

The TiKV server is responsible for storing data. TiKV is a distributed transactional key-value storage engine. **Region** is the basic unit to store data. Each Region stores the data for a particular Key Range which is a left-closed and right-open interval from StartKey to EndKey. Multiple Regions exist in each TiKV node. TiKV APIs provide native support to distributed transactions at the key-value pair level and supports the Snapshot Isolation level isolation by default. This is the core of how TiDB supports distributed transactions at the SQL level. After processing SQL statements, the TiDB server converts the SQL execution plan to an actual call to the TiKV API. Therefore, data is stored in TiKV. All the data in TiKV is automatically maintained in multiple replicas (three replicas by default), so TiKV has native high availability and supports automatic failover.

##### 11.1.1.3.2 TiFlash server

The TiFlash Server is a special type of storage server. Unlike ordinary TiKV nodes, TiFlash stores data by column, mainly designed to accelerate analytical processing.

#### 11.1.2 TiDB Storage

This document introduces some design ideas and key concepts of [TiKV](#).

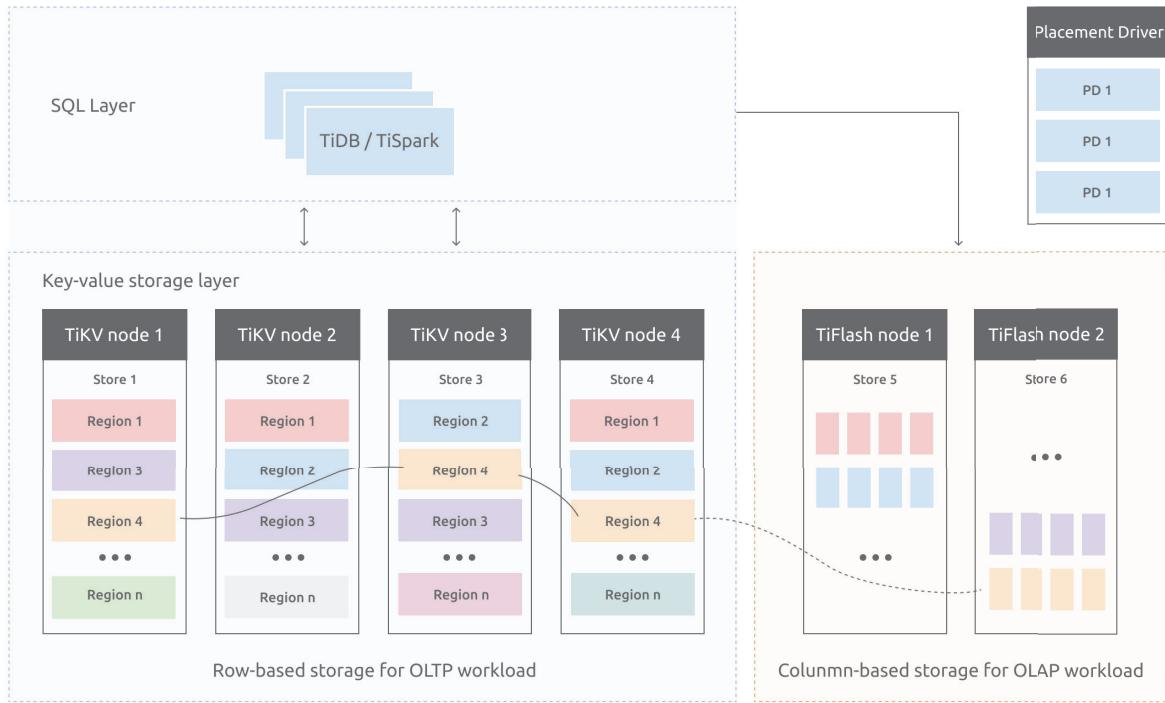


Figure 170: storage-architecture

#### 11.1.2.1 Key-Value pairs

The first thing to decide for a data storage system is the data storage model, that is, in what form the data is saved. TiKV's choice is the Key-Value model and provides an ordered traversal method. There are two key points for TiKV data storage model:

- This is a huge Map (similar to `std::Map` in C++) , which stores Key-Value pairs.
- The Key-Value pair in the Map is ordered according to Keys' binary order, which means you can Seek the position of a particular Key and then call the Next method to get the Key-Value pairs larger than this Key in incremental order.

Note that the KV storage model for TiKV described in this document has nothing to do with tables of SQL. This document does not discuss any concepts related to SQL and only focuses on how to implement a high-performance, high-reliability, distributed Key-Value storage such as TiKV.

#### 11.1.2.2 Local storage (RocksDB)

For any persistent storage engine, data is eventually saved on disk, and TiKV is no exception. TiKV does not write data directly on the disk, but stores data in RocksDB,

which is responsible for the data storage. The reason is that it costs a lot to develop a standalone storage engine, especially a high-performance standalone engine that requires careful optimization.

RocksDB is an excellent standalone storage engine open-sourced by Facebook. This engine can meet various requirements of TiKV for a single engine. Here, you can simply consider RocksDB as a single persistent Key-Value Map.

#### 11.1.2.3 Raft protocol

What's more, the implementation of TiKV faces a more difficult thing: to secure data safety in case a single machine fails.

A simple way is to replicate data to multiple machines, so that even if one machine fails, the replicas on other machines are still available. In other words, you need a data replication scheme that is reliable, efficient, and able to handle the situation of a failed replica. All of these are made possible by the Raft algorithm.

Raft is a consensus algorithm. This document only briefly introduces Raft. For more details, you can see [In Search of an Understandable Consensus Algorithm](#). The Raft has several important features:

- Leader election
- Membership changes (such as adding replicas, deleting replicas, transferring leaders, and so on)
- Log replication

TiKV uses Raft to perform data replication. Each data change will be recorded as a Raft log. Through Raft log replication, data is safely and reliably replicated to multiple nodes of the Raft group. However, according to Raft protocol, successful writes only need that data is replicated to the majority of nodes.

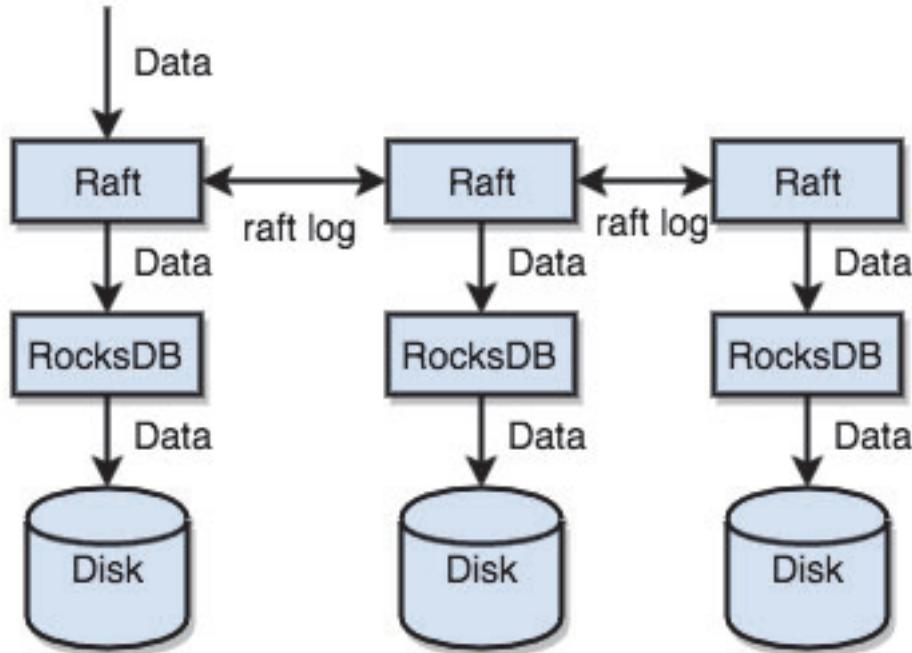


Figure 171: Raft in TiDB

In summary, TiKV can quickly store data on disk via the standalone machine RocksDB, and replicate data to multiple machines via Raft in case of machine failure. Data is written through the interface of Raft instead of to RocksDB. With the implementation of Raft, TiKV becomes a distributed Key-Value storage. Even with a few machine failures, TiKV can automatically complete replicas by virtue of the native Raft protocol, which does not impact the application.

#### 11.1.2.4 Region

To make it easy to understand, let's assume that all data only has one replica. As mentioned earlier, TiKV can be regarded as a large, orderly KV Map, so data is distributed across multiple machines in order to achieve horizontal scalability. For a KV system, there are two typical solutions to distributing data across multiple machines:

- Hash: Create Hash by Key and select the corresponding storage node according to the Hash value.
- Range: Divide ranges by Key, where a segment of serial Key is stored on a node.

TiKV chooses the second solution that divides the whole Key-Value space into a series of consecutive Key segments. Each segment is called a Region. There is a size limit for each Region to store data (the default value is 96 MB and the size can be configured). Each Region can be described by [StartKey, EndKey), a left-closed and right-open interval.

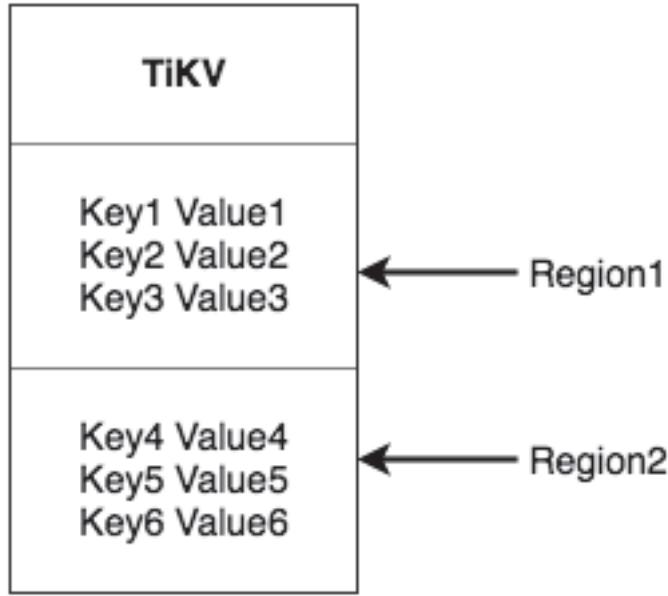


Figure 172: Region in TiDB

Note that the Region here has nothing to do with the table in SQL. In this document, forget about SQL and focus on KV for now. After dividing data into Regions, TiKV will perform two important tasks:

- Distributing data to all nodes in the cluster and use Region as the basic unit. Try its best to ensure that the number of Regions on each node is roughly similar.
- Performing Raft replication and membership management in Region.

These two tasks are very important and will be introduced one by one.

- First, data is divided into many Regions according to Key, and the data for each Region is stored on only one node (ignoring multiple replicas). The TiDB system has a PD component that is responsible for spreading Regions as evenly as possible across all nodes in the cluster. In this way, on one hand, the storage capacity is scaled horizontally (Regions on the other nodes are automatically scheduled to the newly added node); on the other hand, load balancing is achieved (the situation where one node has a lot of data while the others have little will not occur).

At the same time, in order to ensure that the upper client can access the needed data, there is a component (PD) in the system to record the distribution of Regions on the node, that is, the exact Region of a Key and the node of that Region placed through any Key.

- For the second task, TiKV replicates data in Regions, which means that data in one Region will have multiple replicas with the name “Replica”. Multiple Replicas of a

Regions are stored on different nodes to form a Raft Group, which is kept consistent through the Raft algorithm.

One of the Replicas serves as the Leader of the Group and others as the Followers. By default, all reads and writes are processed through the Leader, where reads are done and writes are replicated to followers. The following diagram shows the whole picture about Region and Raft group.

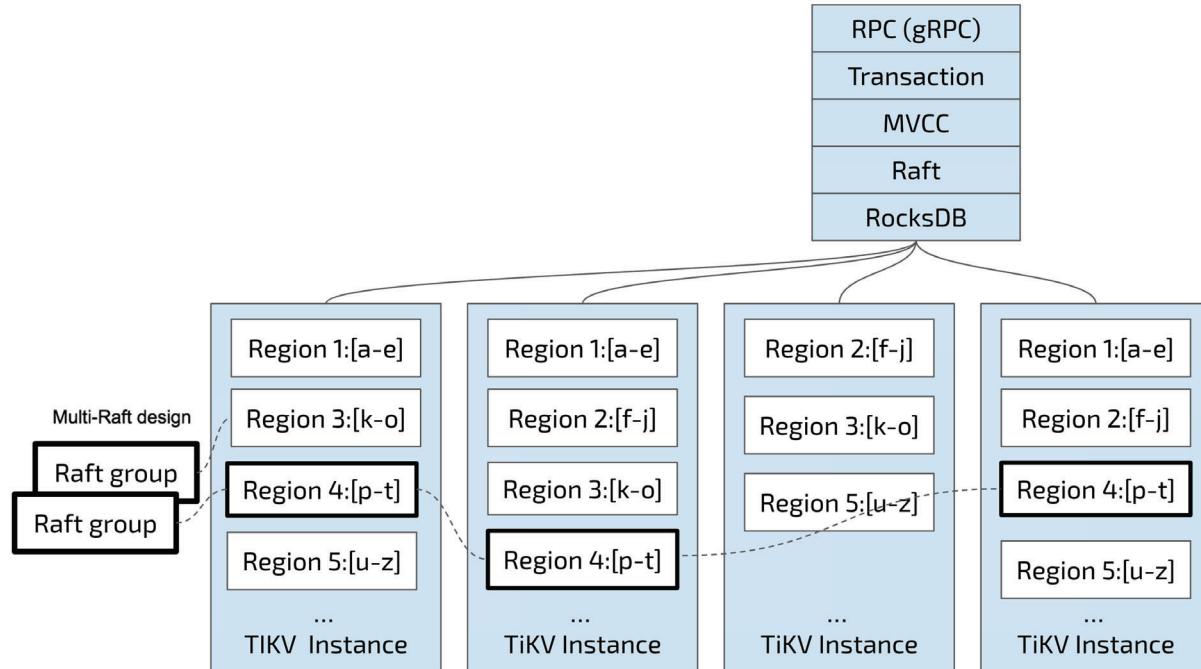


Figure 173: TiDB Storage

As we distribute and replicate data in Regions, we have a distributed Key-Value system that, to some extent, has the capability of disaster recovery. You no longer need to worry about the capacity, or disk failure and data loss.

#### 11.1.2.5 MVCC

Many databases implement multi-version concurrency control (MVCC), and TiKV is no exception. Imagine the situation where two clients modify the value of a Key at the same time. Without MVCC, the data needs to be locked. In a distributed scenario, it might cause performance and deadlock problems. TiKV's MVCC implementation is achieved by appending a version number to Key. In short, without MVCC, TiKV's data layout can be seen as:

Key1 → Value
Key2 → Value

.....  
KeyN -> Value

With MVCC, the key array of TiKV is like this:

```
Key1_Version3 -> Value
Key1_Version2 -> Value
Key1_Version1 -> Value
.....
Key2_Version4 -> Value
Key2_Version3 -> Value
Key2_Version2 -> Value
Key2_Version1 -> Value
.....
KeyN_Version2 -> Value
KeyN_Version1 -> Value
.....
```

Note that for multiple versions of the same Key, versions with larger numbers are placed first (see the [Key-Value](#) section where Keys are arranged in order), so that when you obtain Value through Key + Version, the Key of MVCC can be constructed with Key and Version, which is `Key_Version`. Then you can directly locate the first position greater than or equal to this `Key_Version` through RocksDB's `SeekPrefix(Key_Version)` API.

#### 11.1.2.6 Distributed ACID transaction

Transaction of TiKV adopts the model used by Google in BigTable: [Percolator](#). TiKV's implementation is inspired by this paper, with a lot of optimizations. See [transaction overview](#) for details.

#### 11.1.3 TiDB Computing

Based on the distributed storage provided by TiKV, TiDB builds the computing engine that combines great capability of transactional processing with that of data analysis. This document starts by introducing a data mapping algorithm that maps data from TiDB database tables to (Key, Value) key-value pairs in TiKV, then introduces how TiDB manages metadata, and finally illustrates the architecture of the TiDB SQL layer.

For the storage solution on which the computing layer is dependent, this document only introduces the row-based storage structure of TiKV. For OLAP services, TiDB introduces a column-based storage solution [TiFlash](#) as a TiKV extension.

##### 11.1.3.1 Mapping table data to Key-Value

This section describes the scheme for mapping data to (Key, Value) key-value pairs in TiDB. Data to be mapped here includes the following two types:

- Data of each row in the table, hereinafter referred to as table data.
- Data of all indexes in the table, hereinafter referred to as index data.

#### 11.1.3.1.1 Mapping of table data to Key-Value

In a relational database, a table might have many columns. To map the data of each column in a row to a (Key, Value) key-value pair, you need to consider how to construct the Key. First of all, in OLTP scenarios, there are many operations such as adding, deleting, changing, and searching for data on a single or multiple rows, which needs the database to read a row of data quickly. Therefore, each key should have a unique ID (either explicit or implicit) to make it quick to locate. Then, many OLAP queries require a full table scan. If you can encode the keys of all rows in a table into a range, the whole table can be efficiently scanned by range queries.

Based on the considerations above, the mapping of table data to Key-Value in TiDB is designed as follows:

- To ensure that data from the same table is kept together for easy searching, TiDB assigns a table ID to each table represented by `TableID`. Table ID is an integer that is unique throughout the cluster.
- TiDB assigns a row ID, represented by `RowID`, to each row of data in the table. The row ID is also an integer, unique within the table. For row ID, TiDB has made a small optimization: if a table has an integer type primary key, TiDB uses the value of this primary key as the row ID.

Each row of data is encoded as a (Key, Value) key-value pair according to the following rule:

```
Key: tablePrefix{TableID}_recordPrefixSep{RowID}
Value: [col1, col2, col3, col4]
```

`tablePrefix` and `recordPrefixSep` are both special string constants used to distinguish other data in Key space. The exact values of the string constants are introduced in [Summary of mapping relationships](#).

#### 11.1.3.1.2 Mapping of indexed data to Key-Value

TiDB supports both primary keys and secondary indexes (both unique and non-unique indexes). Similar to the table data mapping scheme, TiDB assigns an index ID to each index of the table represented by `IndexID`.

For primary keys and unique indexes, it is needed to quickly locate the corresponding `RowID` based on the key-value pair, so such a key-value pair is encoded as follows.

```
Key: tablePrefix{tableID}_indexPrefixSep{indexID}_indexedColumnsValue
Value: RowID
```

For ordinary secondary indexes that do not need to satisfy the uniqueness constraint, a single key might correspond to multiple rows. It needs to query corresponding RowID according to the range of keys. Therefore, the key-value pair must be encoded according to the following rule:

```
Key:  tablePrefix{TableID}_indexPrefixSep{IndexID}_indexedColumnsValue_{  
      ↪ RowID}  
Value: null
```

#### 11.1.3.1.3 Summary of mapping relationships

`tablePrefix`, `recordPrefixSep`, and `indexPrefixSep` in all of the above encoding rules are string constants that are used to distinguish a KV from other data in the Key space, which are defined as follows:

```
tablePrefix  = []byte{'t'}  
recordPrefixSep = []byte{'r'}  
indexPrefixSep = []byte{'i'}
```

Also note that in the above encoding schemes, no matter table data or index data key encoding scheme, all rows in a table have the same key prefix, and all data of an index also has the same prefix. Data with the same prefixes are thus arranged together in TiKV's key space. Therefore, by carefully designing the encoding scheme of the suffix part to ensure that the pre-encoding and post-encoding comparisons remain the same, the table data or index data can be stored in TiKV in an ordered manner. Using this encoding scheme, all row data in a table is arranged orderly by RowID in the TiKV's key space, and the data of a particular index is also arranged sequentially in the key space according to the specific value of the index data (`indexedColumnsValue`).

#### 11.1.3.1.4 Example of Key-Value mapping relationship

This section shows a simple example for you to understand the Key-Value mapping relationship of TiDB. Suppose the following table exists in TiDB.

```
CREATE TABLE User (  
    ID int,  
    Name varchar(20),  
    Role varchar(20),  
    Age int,  
    PRIMARY KEY (ID),  
    KEY idxAge (Age)  
) ;
```

Suppose there are 3 rows of data in the table.

```
1, "TiDB", "SQL Layer", 10  
2, "TiKV", "KV Engine", 20
```

3, "PD", "Manager", 30
------------------------

Each row of data is mapped to a (Key, Value) key-value pair, and the table has an `int` type primary key, so the value of `RowID` is the value of this primary key. Suppose the table's `TableID` is 10, and then its table data stored on TiKV is:

<pre>t10_r1 --&gt; ["TiDB", "SQL Layer", 10] t10_r2 --&gt; ["TiKV", "KV Engine", 20] t10_r3 --&gt; ["PD", "Manager", 30]</pre>
--

In addition to the primary key, the table has a non-unique ordinary secondary index, `idxAge`. Suppose the `IndexID` is 1, and then its index data stored on TiKV is:

<pre>t10_i1_10_1 --&gt; null t10_i1_20_2 --&gt; null t10_i1_30_3 --&gt; null</pre>
--

The above example shows the mapping rule from a relational model to a Key-Value model in TiDB, and the consideration behind this mapping scheme.

### 11.1.3.2 Metadata management

Each database and table in TiDB has metadata that indicates its definition and various attributes. This information also needs to be persisted, and TiDB stores this information in TiKV as well.

Each database or table is assigned a unique ID. As the unique identifier, when table data is encoded to Key-Value, this ID is encoded in the Key with the `m_` prefix. This constructs a key-value pair with the serialized metadata stored in it.

In addition, TiDB also uses a dedicated (Key, Value) key-value pair to store the latest version number of structure information of all tables. This key-value pair is global, and its version number is increased by 1 each time the state of the DDL operation changes. TiDB stores this key-value pair persistently in the PD server with the key of `/tidb/ddl → /global_schema_version`, and Value is the version number value of the `int64` type. Meanwhile, because TiDB applies schema changes online, it keeps a background thread that constantly checks whether the version number of the table structure information stored in the PD server changes. This thread also ensures that the changes of version can be obtained within a certain period of time.

### 11.1.3.3 SQL layer overview

TiDB's SQL layer, TiDB Server, translates SQL statements into Key-Value operations, forwards the operations to TiKV, the distributed Key-Value storage layer, assembles the results returned by TiKV, and finally returns the query results to the client.

The nodes at this layer are stateless. These nodes themselves do not store data and are completely equivalent.

#### 11.1.3.3.1 SQL computing

The simplest solution to SQL computing is the **mapping of table data to Key-Value** as described in the previous section, which maps SQL queries to KV queries, acquires the corresponding data through the KV interface, and performs various computations.

For example, to execute the `select count(*) from user where name = "TiDB"` SQL statement, TiDB needs to read all data in the table, then checks whether the `name` field is `TiDB`, and if so, returns this row. The process is as follows:

1. Construct the Key Range: all RowID in a table are in  $[0, \text{MaxInt64})$  range. According to the row data Key encoding rule, using 0 and `MaxInt64` can construct a  $[\text{StartKey} \rightarrow, \text{EndKey})$  range that is left-closed and right-open.
2. Scan Key Range: read the data in TiKV according to the key range constructed above.
3. Filter data: for each row of data read, calculate the `name = "TiDB"` expression. If the result is `true`, return to this row. If not, skip this row.
4. Calculate Count(\*): for each row that meets the requirements, add up to the result of `Count(*)`.

The entire process is illustrated as follows:

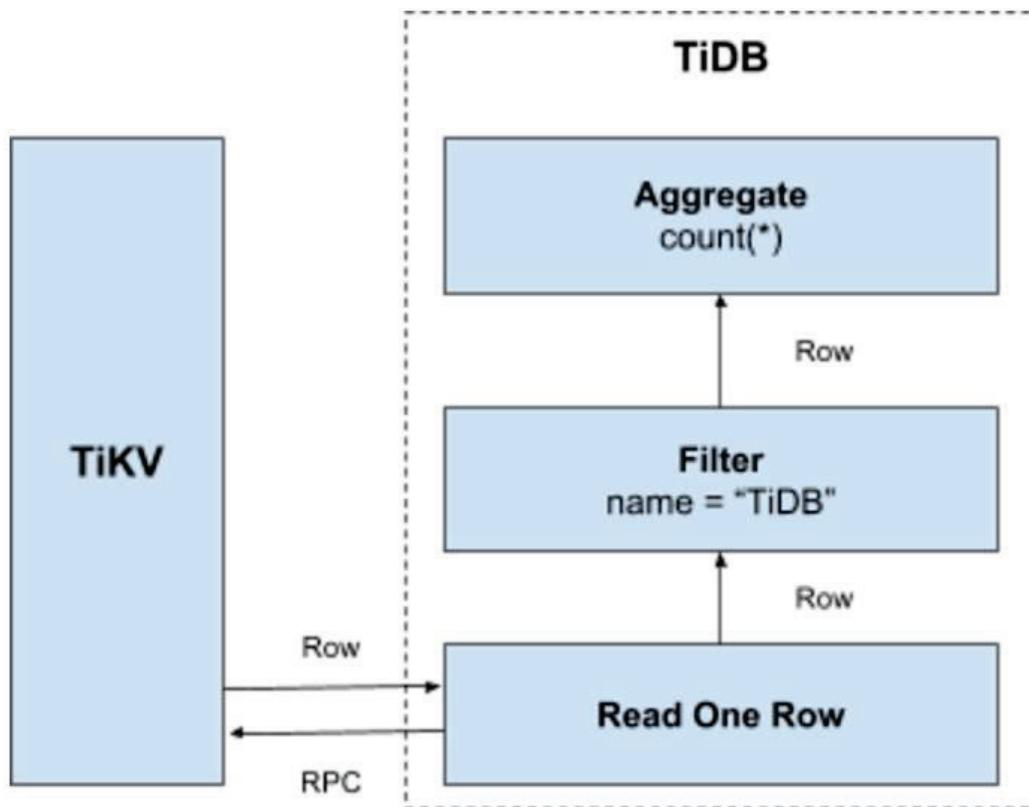


Figure 174: naive sql flow

This solution is intuitive and feasible, but has some obvious problems in a distributed database scenario:

- As the data is being scanned, each row is read from TiKV via a KV operation with at least one RPC overhead, which can be very high if there is a large amount of data to be scanned.
- It is not applicable to all rows. Data that does not meet the conditions does not need to be read.
- From the returned result of this query, only the number of rows that match the requirements is needed, not the value of those rows.

#### 11.1.3.3.2 Distributed SQL operations

To solve the problems above, the computation should be as close to the storage node as possible to avoid a large number of RPC callings. First of all, the SQL predicate condition `name = "TiDB"` should be pushed down to the storage node for computation, so that only valid rows are returned, which avoids meaningless network transfers. Then, the aggregation function `Count(*)` can also be pushed down to the storage nodes for pre-aggregation, and each node only has to return a result of `Count(*)`. The SQL layer will sum up the `Count(*)` results returned by each node.

The following image shows how data returns layer by layer:

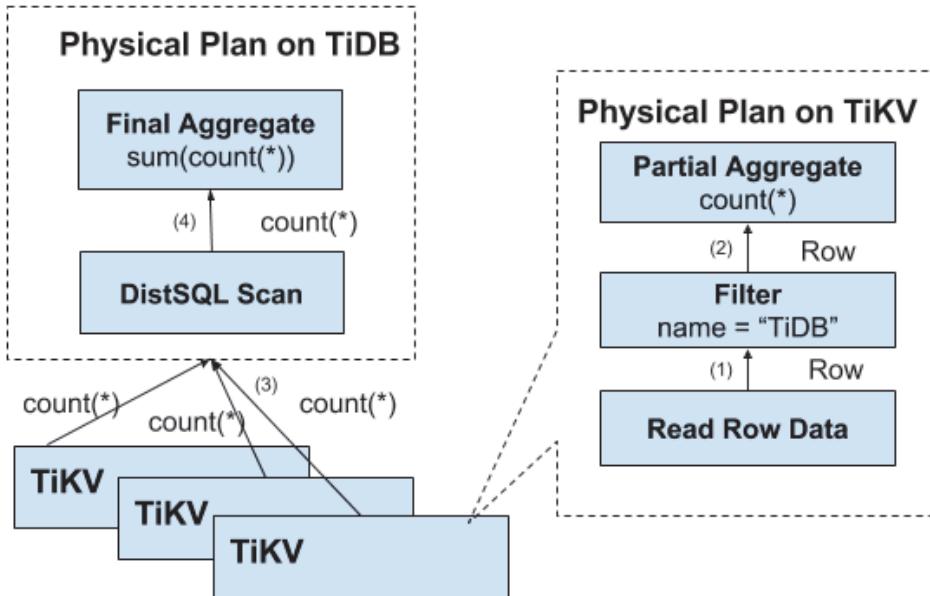


Figure 175: dist sql flow

#### 11.1.3.3.3 Architecture of SQL layer

The previous sections introduce some functions of the SQL layer and I hope you have a basic understanding of how SQL statements are handled. In fact, TiDB's SQL layer is much more complicated, with many modules and layers. The following diagram lists the important modules and calling relationships:

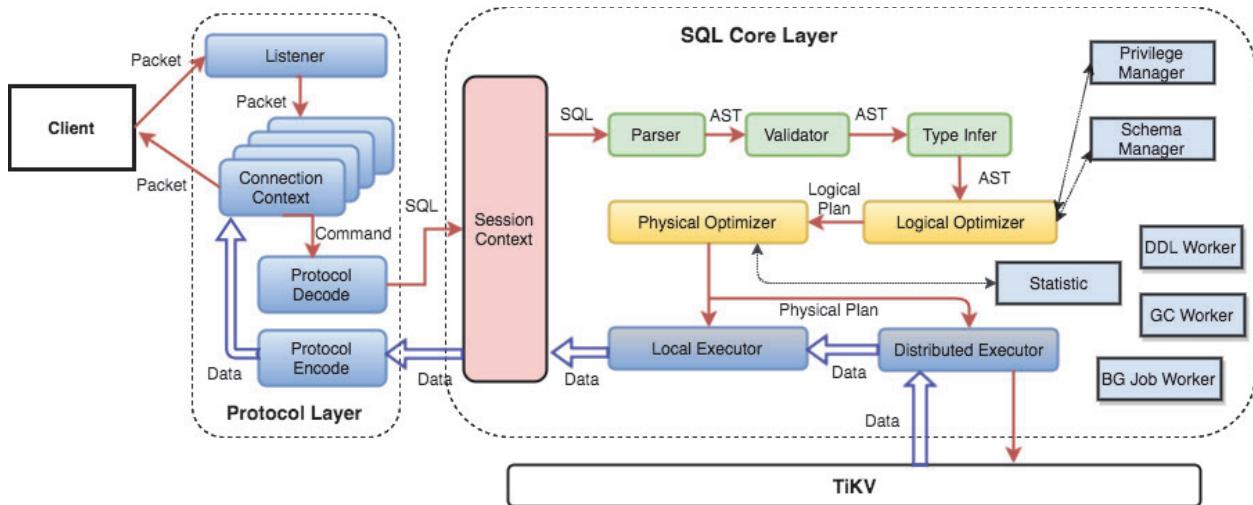


Figure 176: tidb sql layer

The user's SQL request is sent to TiDB Server either directly or via Load Balancer. TiDB Server will parse MySQL Protocol Packet, get the content of requests, parse the SQL request syntactically and semantically, develop and optimize query plans, execute a query plan, get and process the data. All data is stored in the TiKV cluster, so in this process, TiDB Server needs to interact with TiKV and get the data. Finally, TiDB Server needs to return the query results to the user.

#### 11.1.4 TiDB Scheduling

The Placement Driver (PD) works as the manager in a TiDB cluster, and it also schedules Regions in the cluster. This article introduces the design and core concepts of the PD scheduling component.

##### 11.1.4.1 Scheduling situations

TiKV is the distributed key-value storage engine used by TiDB. In TiKV, data is organized as Regions, which are replicated on several stores. In all replicas, a leader is responsible for reading and writing, and followers are responsible for replicating Raft logs from the leader.

Now consider about the following situations:

- To utilize storage space in a high-efficient way, multiple Replicas of the same Region need to be properly distributed on different nodes according to the Region size;
- For multiple data center topologies, one data center failure only causes one replica to fail for all Regions;
- When a new TiKV store is added, data can be rebalanced to it;
- When a TiKV store fails, PD needs to consider:

- Recovery time of the failed store.
  - \* If it's short (for example, the service is restarted), whether scheduling is necessary or not.
  - \* If it's long (for example, disk fault, data is lost, etc.), how to do scheduling.
- Replicas of all Regions.
  - \* If the number of replicas is not enough for some Regions, PD needs to complete them.
  - \* If the number of replicas is more than expected (for example, the failed store re-joins into the cluster after recovery), PD needs to delete them.
- Read/Write operations are performed on leaders, which can not be distributed only on a few individual stores;
- Not all Regions are hot, so loads of all TiKV stores need to be balanced;
- When Regions are in balancing, data transferring utilizes much network/disk traffic and CPU time, which can influence online services.

These situations can occur at the same time, which makes it harder to resolve. Also, the whole system is changing dynamically, so a scheduler is needed to collect all information about the cluster, and then adjust the cluster. So, PD is introduced into the TiDB cluster.

#### 11.1.4.2 Scheduling requirements

The above situations can be classified into two types:

1. A distributed and highly available storage system must meet the following requirements:
  - The right number of replicas.
  - Replicas need to be distributed on different machines according to different topologies.
  - The cluster can perform automatic disaster recovery from TiKV peers' failure.
2. A good distributed system needs to have the following optimizations:
  - All Region leaders are distributed evenly on stores;
  - Storage capacity of all TiKV peers are balanced;
  - Hot spots are balanced;
  - Speed of load balancing for the Regions needs to be limited to ensure that online services are stable;
  - Maintainers are able to take peers online/offline manually.

After the first type of requirements is satisfied, the system will be failure tolerable. After the second type of requirements is satisfied, resources will be utilized more efficiently and the system will have better scalability.

To achieve the goals, PD needs to collect information firstly, such as state of peers, information about Raft groups and the statistics of accessing the peers. Then we need to specify some strategies for PD, so that PD can make scheduling plans from these information and strategies. Finally, PD distributes some operators to TiKV peers to complete scheduling plans.

#### 11.1.4.3 Basic scheduling operators

All scheduling plans contain three basic operators:

- Add a new replica
- Remove a replica
- Transfer a Region leader between replicas in a Raft group

They are implemented by the Raft commands `AddReplica`, `RemoveReplica`, and `TransferLeader`.

#### 11.1.4.4 Information collection

Scheduling is based on information collection. In short, the PD scheduling component needs to know the states of all TiKV peers and all Regions. TiKV peers report the following information to PD:

- State information reported by each TiKV peer:

Each TiKV peer sends heartbeats to PD periodically. PD not only checks whether the store is alive, but also collects `StoreState` in the heartbeat message. `StoreState` includes:

- Total disk space
- Available disk space
- The number of Regions
- Data read/write speed
- The number of snapshots that are sent/received (The data might be replicated between replicas through snapshots)
- Whether the store is overloaded
- Labels (See [Perception of Topology](#))

- Information reported by Region leaders:

Each Region leader sends heartbeats to PD periodically to report `RegionState`, including:

- Position of the leader itself
- Positions of other replicas
- The number of offline replicas

- data read/write speed

PD collects cluster information by the two types of heartbeats and then makes decision based on it.

Besides, PD can get more information from an expanded interface to make a more precise decision. For example, if a store's heartbeats are broken, PD can't know whether the peer steps down temporarily or forever. It just waits a while (by default 30min) and then treats the store as offline if there are still no heartbeats received. Then PD balances all regions on the store to other stores.

But sometimes stores are manually set offline by a maintainer, so the maintainer can tell PD this by the PD control interface. Then PD can balance all regions immediately.

#### 11.1.4.5 Scheduling strategies

After collecting the information, PD needs some strategies to make scheduling plans.

##### **Strategy 1: The number of replicas of a Region needs to be correct**

PD can know that the replica count of a Region is incorrect from the Region leader's heartbeat. If it happens, PD can adjust the replica count by adding/removing replica(s). The reason for incorrect replica count can be:

- Store failure, so some Region's replica count is less than expected;
- Store recovery after failure, so some Region's replica count could be more than expected;
- `max-replicas` is changed.

##### **Strategy 2: Replicas of a Region need to be at different positions**

Note that here “position” is different from “machine”. Generally PD can only ensure that replicas of a Region are not at a same peer to avoid that the peer's failure causes more than one replicas to become lost. However in production, you might have the following requirements:

- Multiple TiKV peers are on one machine;
- TiKV peers are on multiple racks, and the system is expected to be available even if a rack fails;
- TiKV peers are in multiple data centers, and the system is expected to be available even if a data center fails;

The key to these requirements is that peers can have the same “position”, which is the smallest unit for failure toleration. Replicas of a Region must not be in one unit. So, we can configure `labels` for the TiKV peers, and set `location-labels` on PD to specify which labels are used for marking positions.

### Strategy 3: Replicas need to be balanced between stores

The size limit of a Region replica is fixed, so keeping the replicas balanced between stores is helpful for data size balance.

### Strategy 4: Leaders need to be balanced between stores

Read and write operations are performed on leaders according to the Raft protocol, so that PD needs to distribute leaders into the whole cluster instead of several peers.

### Strategy 5: Hot spots need to be balanced between stores

PD can detect hot spots from store heartbeats and Region heartbeats, so that PD can distribute hot spots.

### Strategy 6: Storage size needs to be balanced between stores

When started up, a TiKV store reports capacity of storage, which indicates the store's space limit. PD will consider this when scheduling.

### Strategy 7: Adjust scheduling speed to stabilize online services

Scheduling utilizes CPU, memory, network and I/O traffic. Too much resource utilization will influence online services. Therefore, PD needs to limit the number of the concurrent scheduling tasks. By default this strategy is conservative, while it can be changed if quicker scheduling is required.

#### 11.1.4.6 Scheduling implementation

PD collects cluster information from store heartbeats and Region heartbeats, and then makes scheduling plans from the information and strategies. Scheduling plans are a sequence of basic operators. Every time PD receives a Region heartbeat from a Region leader, it checks whether there is a pending operator on the Region or not. If PD needs to dispatch a new operator to a Region, it puts the operator into heartbeat responses, and monitors the operator by checking follow-up Region heartbeats.

Note that here “operators” are only suggestions to the Region leader, which can be skipped by Regions. Leader of Regions can decide whether to skip a scheduling operator or not based on its current status.

## 11.2 Key Monitoring Metrics

### 11.2.1 Key Metrics

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For more information, see [TiDB Monitoring Framework Overview](#).

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node\_exporter, Disk Performance, and so on. A lot of metrics are there to help you diagnose.

For routine operations, you can get an overview of the component (PD, TiDB, TiKV) status and the entire cluster from the Overview dashboard, where the key metrics are displayed. This document provides a detailed description of these key metrics.

### 11.2.1.1 Key metrics description

To understand the key metrics displayed on the Overview dashboard, check the following table:

Service Panel Name	Description	Normal Range
Services Services Up	The online nodes number of each service.	
Port		
Status		
PD PD role	The role of the current PD.	
PD Storage capacity	The total storage capacity of the TiDB cluster.	
PD Current storage size	The occupied storage capacity of the TiDB cluster, including the space occupied by TiKV replicas.	
PD Normal stores	The number of nodes in the normal state.	
PD Abnormal stores	The number of nodes in the abnormal state. 0	0
PD Number of Regions	The total number of Regions in the current cluster. Note that the number of Regions has nothing to do with the number of replicas.	
PD 99% completed_cmds_duration	The 99th percentile duration to complete a pd-server request. less than 5ms	less than 5ms
PD Handle_requests_duration	The duration of a PD request. That is the time spent in the network of a PD request.	
PD Region health	The state of each Region. Generally, the number of pending peers is less than 100, and that of the missing peers cannot always be greater than 0.	Generally, the number of pending peers is less than 100, and that of the missing peers cannot always be greater than 0.
PD Hot write Region's leader distribution	The total number of leaders who are the write hotspots on each TiKV instance.	

Service Panel Name	Description	Normal Range
PD Hot read Region's leader distribution	The total number of leaders who are the read hotspots on each TiKV instance.	
PD Region heartbeat report	The count of heartbeats reported to PD per instance.	
PD 99% Region heartbeat latency	The heartbeat latency per TiKV instance (P99).	
TiDB Statement OPS	The number of different types of SQL statements executed per second, which is counted according to <code>SELECT</code> , <code>INSERT</code> , <code>UPDATE</code> , and other types of statements.	
TiDB Duration	The execution time.1. The duration between the time that the client's network request is sent to TiDB and the time that the request is returned to the client after TiDB has executed the request. In general, client requests are sent in the form of SQL statements; however, this duration can include the execution time of commands such as <code>COM_PING</code> , <code>COM_SLEEP</code> , <code>COM_STMT_FETCH</code> , and <code>COM_SEND_LONG_DATA</code> .2. Because TiDB supports Multi-Query, TiDB supports sending multiple SQL statements at one time, such as <code>select 1; select 1; select 1;</code> . In this case, the total execution time of this query includes the execution time of all SQL statements.	
TiDB CPS By Instance	CPS By Instance: the command statistics on each TiDB instance, which is classified according to the success or failure of command execution results.	

Service Panel Name	Description	Normal Range
TiDB Failed Query OPM	The statistics of error types (such as syntax errors and primary key conflicts) based on the errors occurred when executing SQL statements per second on each TiDB instance. The module in which the error occurs and the error code are included.	
TiDB Connection Count	The connection number of each TiDB instance.	
TiDB Memory Usage	The memory usage statistics of each TiDB instance, which is divided into the memory occupied by processes and the memory applied by Golang on the heap.	
TiDB Transaction OPS	The number of transactions executed per second.	
TiDB Transaction Duration	The execution time of a transaction	
TiDB KV Cmd OPS	The number of executed KV commands.	
TiDB KV Cmd Duration 99	The execution time of the KV command.	
TiDB PD TSO OPS	The number of TSO that TiDB obtains from PD per second.	
TiDB PD TSO Wait Duration	The duration that TiDB waits for PD to return TSO.	
TiDB TiClient Region Error OPS	The number of Region related errors returned by TiKV.	
TiDB Lock Resolve OPS	The number of TiDB operations that resolve locks. When TiDB's read or write request encounters a lock, it tries to resolve the lock.	
TiDB KV Backoff OPS	The number of errors returned by TiKV.	
TiKV leader	The number of leaders on each TiKV node.	
TiKV region	The number of Regions on each TiKV node.	
TiKV CPU	The CPU usage ratio on each TiKV node.	
TiKV Memory	The memory usage on each TiKV node.	

Service Panel Name	Description	Normal Range
TiKV store size	The size of storage space used by each TiKV instance.	
TiKV cf size	The size of each column family (CF for short).	
TiKV channel full	The number of “channel full” errors on each TiKV instance.	0
TiKV server report failures	The number of error messages reported by each TiKV instance.	0
TiKV scheduler pending commands	The number of pending commands on each TiKV instance.	
TiKV coprocessor executor count	The number of coprocessor operations received by TiKV per second. Each type of coprocessor is counted separately.	
TiKV coprocessor request duration	The time consumed to process read requests of coprocessor.	
TiKV raft store CPU	The CPU usage ratio of the raftstore thread	The default number of threads is 2 (configured by <code>raftstore.raft-store-pool-size</code> ). A value of over 80% for a single thread indicates that the CPU usage ratio is very high.
TiKV Coprocessor CPU	The CPU usage ratio of the coprocessor thread.	
System Vcores Info	The number of CPU cores.	
System Memory Info	The total memory.	
System CPU Usage Info	The CPU usage ratio, 100% at a maximum.	
System Load [1m] Info	The overload within 1 minute.	
System Memory Info Available	The size of the available memory.	

Service Panel Name	Description	Normal Range
System Network Traffic Info	The statistics of the network traffic.	
System TCP Retrans Info	The frequency of the TOC retransmission.	
System IO Util Info	The disk usage ratio, 100% at a maximum; generally you need to consider adding a new node when the usage ratio is up to 80% ~ 90%.	

### 11.2.1.2 Interface of the Overview dashboard

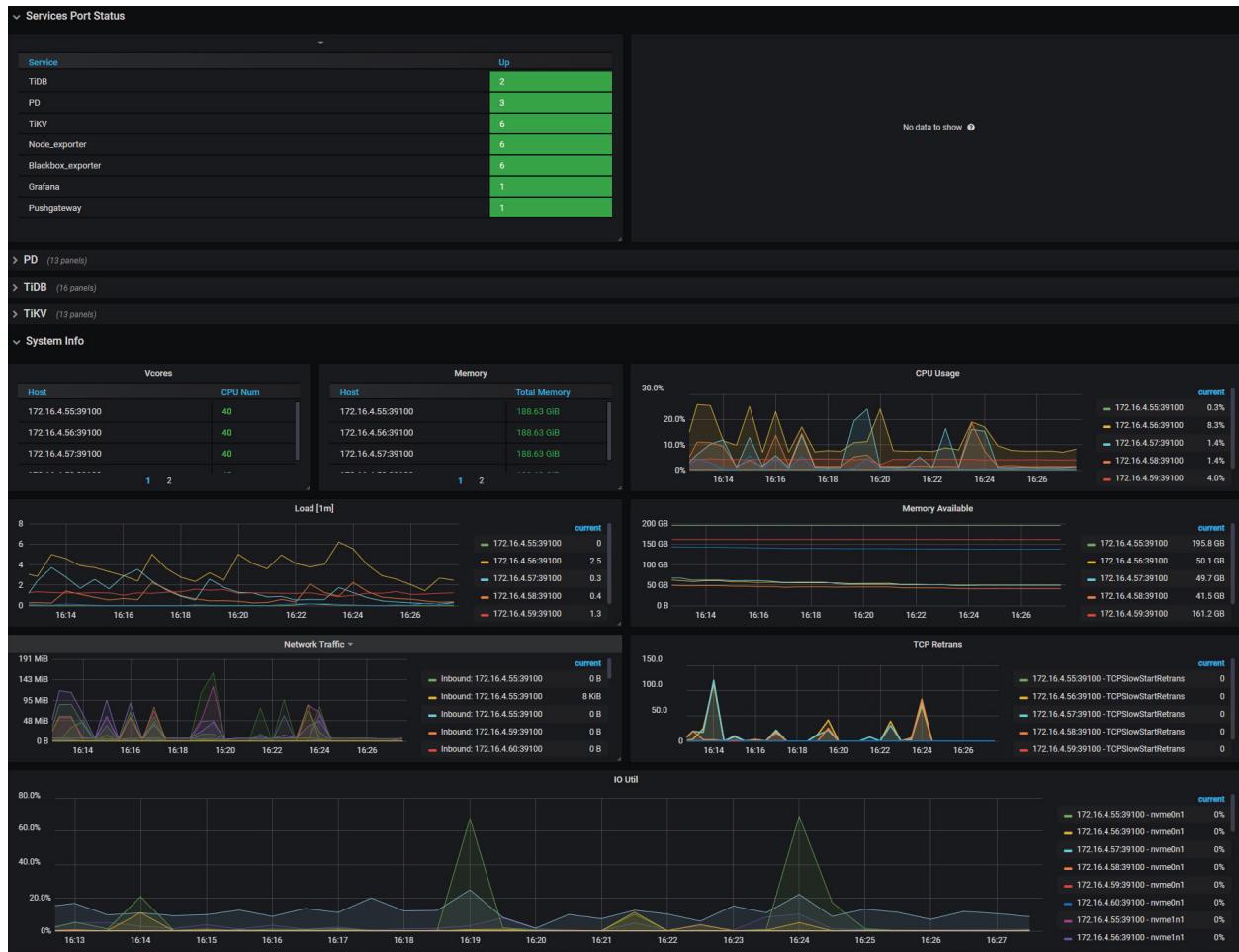


Figure 177: overview

### 11.2.2 TiDB Monitoring Metrics

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For the monitoring architecture, see [TiDB Monitoring Framework Overview](#).

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node\_exporter, Disk Performance, and so on. The TiDB dashboard consists of the TiDB panel and the TiDB Summary panel. The differences between the two panels are different in the following aspects:

- TiDB panel: provides as comprehensive information as possible for troubleshooting cluster anomalies.
- TiDB Summary Panel: extracts parts of the TiDB panel information with which users are most concerned, with some modifications. It provides data (such as QPS, TPS, response delay) that users care about in the daily database operations, which serves as the monitoring information to be displayed or reported.

This document describes some key monitoring metrics displayed on the TiDB dashboard.

#### 11.2.2.1 Key metrics description

To understand the key metrics displayed on the TiDB dashboard, check the following list:

- Query Summary
  - Duration: execution time
    - \* The duration between the time that the client's network request is sent to TiDB and the time that the request is returned to the client after TiDB has executed it. In general, client requests are sent in the form of SQL statements, but can also include the execution time of commands such as `COM_PING`, `COM_SLEEP`, `COM_STMT_FETCH`, and `COM_SEND_LONG_DATA`
    - \* Because TiDB supports Multi-Query, it supports sending multiple SQL statements at one time, such as `select 1; select 1; select 1;`. In this case, the total execution time of this query includes the execution time of all SQL statements
  - Command Per Second: the number of commands processed by TiDB per second, which is classified according to the success or failure of command execution results
  - QPS: the number of SQL statements executed per second on all TiDB instances, which is counted according to `SELECT`, `INSERT`, `UPDATE`, and other types of statements
  - CPS By Instance: the command statistics on each TiDB instance, which is classified according to the success or failure of command execution results

- Failed Query OPM: the statistics of error types (such as syntax errors and primary key conflicts) according to the errors occurred when executing SQL statements per second on each TiDB instance. It contains the module in which the error occurs and the error code
- Slow query: the statistics of the processing time of slow queries (the time cost of the entire slow query, the time cost of Coprocessor, and the waiting time for Coprocessor scheduling). Slow queries are classified into internal and general SQL statements
- Connection Idle Duration: the duration of idle connections
- 999/99/95/80 Duration: the statistics of the execution time for different types of SQL statements (different percentiles)
- Query Detail
  - Duration 80/95/99/999 By Instance: the statistics of the execution time for SQL statements on each TiDB instance (different percentiles)
  - Failed Query OPM Detail: the statistics of error types (such as syntax errors and primary key conflicts) according to the errors occurred when executing SQL statements on each TiDB instance
  - Internal SQL OPS: the internal SQL statements executed per second in the entire TiDB cluster. The internal SQL statements are internally executed and are generally triggered by user SQL statements or internally scheduled tasks.
- Server
  - Uptime: the runtime of each TiDB instance
  - Memory Usage: the memory usage statistics of each TiDB instance, which is divided into the memory occupied by processes and the memory applied by Golang on the heap
  - CPU Usage: the statistics of CPU usage of each TiDB instance
  - Connection Count: the number of clients connected to each TiDB instance
  - Open FD Count: the statistics of opened file descriptors of each TiDB instance
  - Disconnection Count: the number of clients disconnected to each TiDB instance
  - Events OPM: the statistics of key events, such as “start”, “close”, “graceful-shutdown”, “kill”, “hang”, and so on
  - Goroutine Count: the number of Goroutines on each TiDB instance
  - Prepare Statement Count: the number of `Prepare` statements that are executed on each TiDB instance and the total count of them
  - Keep Alive OPM: the number of times that the metrics are refreshed every minute on each TiDB instance. It usually needs no attention.
  - Panic And Critical Error: the number of panics and critical errors occurred in TiDB
  - Time Jump Back OPS: the number of times that the operating system rewinds every second on each TiDB instance
  - Get Token Duration: the time cost of getting Token on each connection
  - Skip Binlog Count: the number of binlog write failures in TiDB
  - Client Data Traffic: data traffic statistics of TiDB and the client

- Transaction

- Transaction OPS: the number of transactions executed per second
- Duration: the execution duration of a transaction
- Transaction Statement Num: the number of SQL statements in a transaction
- Transaction Retry Num: the number of times that a transaction retries
- Session Retry Error OPS: the number of errors encountered during the transaction retry per second. This metric includes two error types: retry failure and exceeding the maximum number of retries
- Commit Token Wait Duration: the wait duration in the flow control queue during the transaction commit. If the wait duration is long, it means that the transaction to commit is too large and the flow is controlled. If the system still has resources available, you can speed up the commit process by increasing the `committer-concurrency` value in the TiDB configuration file
- KV Transaction OPS: the number of transactions executed per second within each TiDB instance
  - \* A user transaction might trigger multiple transaction executions in TiDB, including reading internal metadata, atomic retries of the user transaction, and so on
  - \* TiDB's internally scheduled tasks also operate on the database through transactions, which are also included in this panel
- KV Transaction Duration: the time spent on executing transactions within each TiDB
- Transaction Regions Num: the number of Regions operated in the transaction
- Transaction Write KV Num Rate and Sum: the rate at which KVs are written and the sum of these written KVs in the transaction
- Transaction Write KV Num: the number of KVs operated in the transaction
- Statement Lock Keys: the number of locks for a single statement
- Send HeartBeat Duration: the duration for the transaction to send heartbeats
- Transaction Write Size Bytes Rate and sum: the rate at which bytes are written and the sum of these written bytes in the transaction
- Transaction Write Size Bytes: the size of the data written in the transaction
- Acquire Pessimistic Locks Duration: the time consumed by adding locks
- TTL Lifetime Reach Counter: the number of transactions that reach the upper limit of TTL. The default value of the TTL upper limit is 1 hour. It means that 1 hour has passed since the first lock of a pessimistic transaction or the first prewrite of an optimistic transaction. The default value of the upper limit of TTL is 1 hour. The upper limit of TTL life can be changed by modifying `max-txn-TTL` in the TiDB configuration file
- Load Safepoint OPS: the number of times that `Safepoint` is loaded. `Safepoint` is to ensure that the data before `Safepoint` is not read when the transaction reads data, thus ensuring data safety. The data before `Safepoint` might be cleaned up by the GC
- Pessimistic Statement Retry OPS: the number of retry attempts for pessimistic statements. When the statement tries to add lock, it might encounter a write

conflict. At this time, the statement will acquire a new snapshot and add lock again

- Transaction Types Per Seconds: the number of transactions committed per second using the two-phase commit (2PC), async commit, and one-phase commit (1PC) mechanisms, including both success and failure transactions
- Executor
  - Parse Duration: the statistics of the parsing time of SQL statements
  - Compile Duration: the statistics of the time of compiling the parsed SQL AST to the execution plan
  - Execution Duration: the statistics of the execution time for SQL statements
  - Expensive Executor OPS: the statistics of the operators that consume many system resources per second, including `Merge Join`, `Hash Join`, `Index Look Up` ↪ `Join`, `Hash Agg`, `Stream Agg`, `Sort`, `TopN`, and so on
  - Queries Using Plan Cache OPS: the statistics of queries using the Plan Cache per second
- Distsql
  - Distsql Duration: the processing time of Distsql statements
  - Distsql QPS: the statistics of Distsql statements
  - Distsql Partial QPS: the number of Partial results every second
  - Scan Keys Num: the number of keys that each query scans
  - Scan Keys Partial Num: the number of keys that each Partial result scans
  - Partial Num: the number of Partial results for each SQL statement
- KV Errors
  - KV Backoff Duration: the total duration that a KV retry request lasts. TiDB might encounter an error when sending a request to TiKV. TiDB has a retry mechanism for every request to TiKV. This KV Backoff Duration item records the total time of a request retry.
  - TiClient Region Error OPS: the number of Region related error messages returned by TiKV
  - KV Backoff OPS: the number of error messages returned by TiKV
  - Lock Resolve OPS: the number of TiDB operations to resolve locks. When TiDB's read or write request encounters a lock, it tries to resolve the lock
  - Other Errors OPS: the number of other types of errors, including clearing locks and updating `SafePoint`
- KV Request
  - KV Request OPS: the execution times of a KV request, displayed according to TiKV
  - KV Request Duration 99 by store: the execution time of a KV request, displayed according to TiKV
  - KV Request Duration 99 by type: the execution time of a KV request, displayed according to the request type

- PD Client
  - PD Client CMD OPS: the statistics of commands executed by PD Client per second
  - PD Client CMD Duration: the time it takes for PD Client to execute commands
  - PD Client CMD Fail OPS: the statistics of failed commands executed by PD Client per second
  - PD TSO OPS: the number of TSO that TiDB obtains from PD per second
  - PD TSO Wait Duration: the time that TiDB waits for PD to return TSO
  - PD TSO RPC duration: the duration from the time that TiDB sends request to PD (to get TSO) to the time that TiDB receives TSO
  - Start TSO Wait Duration: the duration from the time that TiDB sends request to PD (to get `start TSO`) to the time that TiDB receives `start TSO`
- Schema Load
  - Load Schema Duration: the time it takes TiDB to obtain the schema from TiKV
  - Load Schema OPS: the statistics of the schemas that TiDB obtains from TiKV per second
  - Schema Lease Error OPM: the Schema Lease errors include two types: `change` and `outdate`. `change` means that the schema has changed, and `outdate` means that the schema cannot be updated, which is a more serious error and triggers an alert.
  - Load Privilege OPS: the statistics of the number of privilege information obtained by TiDB from TiKV per second
- DDL
  - DDL Duration 95: 95% quantile of DDL statement processing time
  - Batch Add Index Duration 100: statistics of the maximum time spent by each Batch on creating an index
  - DDL Waiting Jobs Count: the number of DDL tasks that are waiting
  - DDL META OPM: the number of times that a DDL obtains META every minute
  - DDL Worker Duration 99: 99% quantile of the execution time of each DDL worker
  - Deploy Syncer Duration: the time consumed by Schema Version Syncer initialization, restart, and clearing up operations
  - Owner Handle Syncer Duration: the time that it takes the DDL Owner to update, obtain, and check the Schema Version
  - Update Self Version Duration: the time consumed by updating the version information of Schema Version Syncer
  - DDL OPM: the number of DDL executions per second
  - DDL Add Index Progress In Percentage: the progress of adding an index
- Statistics
  - Auto Analyze Duration 95: the time consumed by automatic `ANALYZE`
  - Auto Analyze QPS: the statistics of automatic `ANALYZE`
  - Stats Inaccuracy Rate: the information of the statistics inaccuracy rate

- Pseudo Estimation OPS: the number of the SQL statements optimized using pseudo statistics
- Dump Feedback OPS: the number of stored statistical feedbacks
- Store Query Feedback QPS: the number of operations per second to store the feedback information of the union query, which is performed in TiDB memory
- Significant Feedback: the number of significant feedback pieces that update the statistics information
- Update Stats OPS: the number of operations of updating statistics with feedback
- Fast Analyze Status 100: the status for quickly collecting statistical information
- Owner
  - New ETCD Session Duration 95: the time it takes to create a new etcd session. TiDB connects to etcd in PD through etcd client to save/read some metadata information. This records the time spent creating the session
  - Owner Watcher OPS: the number of Goroutine operations per second of DDL owner watch PD's etcd metadata
- Meta
  - AutoID QPS: AutoID related statistics, including three operations (global ID allocation, a single table AutoID allocation, a single table AutoID Rebase)
  - AutoID Duration: the time consumed by AutoID related operations
  - Region Cache Error OPS: the number of errors encountered per second by the cached Region information in TiDB
  - Meta Operations Duration 99: the latency of Meta operations
- GC
  - Worker Action OPM: the number of GC related operations, including `run_job`, `resolve_lock`, and `delete\range`
  - Duration 99: the time consumed by GC related operations
  - Config: the configuration of GC data life time and GC running interval
  - GC Failure OPM: the number of failed GC related operations
  - Delete Range Failure OPM: the number of times the `Delete Range` has failed
  - Too Many Locks Error OPM: the number of the error that GC clears up too many locks
  - Action Result OPM: the number of results of GC-related operations
  - Delete Range Task Status: the task status of `Delete Range`, including completion and failure
  - Push Task Duration 95: the time spent pushing GC subtasks to GC workers
- Batch Client
  - Pending Request Count by TiKV: the number of Batch messages that are pending processing
  - Batch Client Unavailable Duration 95: the unavailable time of the Batch client
  - No Available Connection Counter: the number of times the Batch client cannot find an available link

### 11.2.3 Key Monitoring Metrics of PD

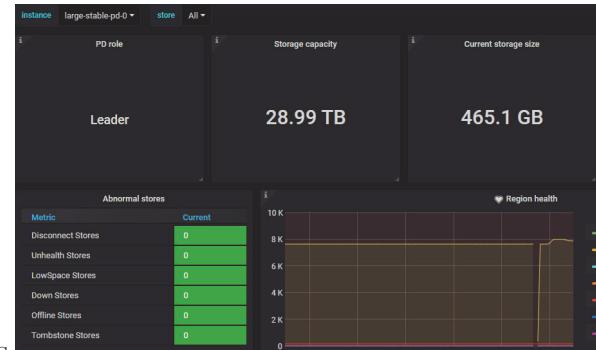
If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For more information, see [Overview of the Monitoring Framework](#).

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node\_exporter, Disk Performance, and so on. A lot of metrics are there to help you diagnose.

You can get an overview of the component PD status from the PD dashboard, where the key metrics are displayed. This document provides a detailed description of these key metrics.

The following is the description of PD Dashboard metrics items:

- PD role: The role of the current PD instance
- Storage capacity: The total storage capacity for this TiDB cluster
- Current storage size: The storage size that is currently used by the TiDB cluster
- Current storage usage: The current storage usage rate
- Normal stores: The count of healthy storage instances
- Number of Regions: The total count of cluster Regions
- Abnormal stores: The count of unhealthy stores. The normal value is 0. If the number is bigger than 0, it means at least one instance is abnormal.
- Region health: The health status of Regions indicated via the count of unusual Regions including pending peers, down peers, extra peers, offline peers, missing peers, learner peers and incorrect namespaces. Generally, the number of pending peers should be less than 100. The missing peers should not be persistently greater than 0. If many empty Regions exist, enable Region Merge in time.



- Current peer count: The current count of all cluster peers

#### 11.2.3.1 Key metrics description

#### 11.2.3.2 Cluster

- PD scheduler config: The list of PD scheduler configurations
- Cluster ID: The unique identifier of the cluster

- Current TSO: The physical part of current allocated TSO
- Current ID allocation: The maximum allocatable ID for new store/peer
- Region label isolation level: The number of Regions in different label levels
- Label distribution: The distribution status of the labels in the cluster

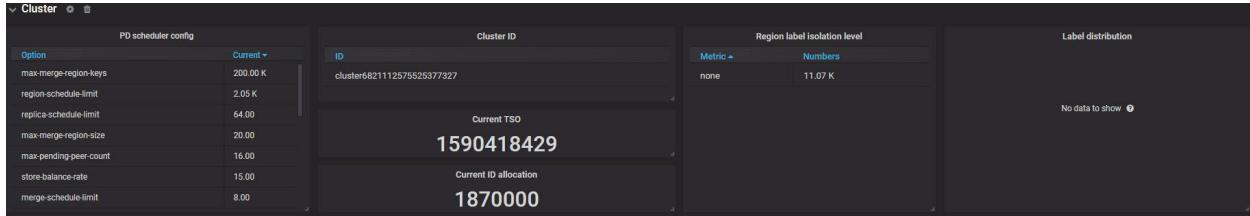


Figure 178: PD Dashboard - Cluster metrics

### 11.2.3.3 Operator

- Schedule operator create: The number of newly created operators per type
- Schedule operator check: The number of checked operator per type. It mainly checks whether the current step is finished; if yes, it returns the next step to be executed
- Schedule operator finish: The number of finished operators per type
- Schedule operator timeout: The number of timeout operators per type
- Schedule operator replaced or canceled: The number of replaced or canceled operators per type
- Schedule operators count by state: The number of operators per state
- Operator finish duration: The maximum duration of finished operators
- Operator step duration: The maximum duration of finished operator steps

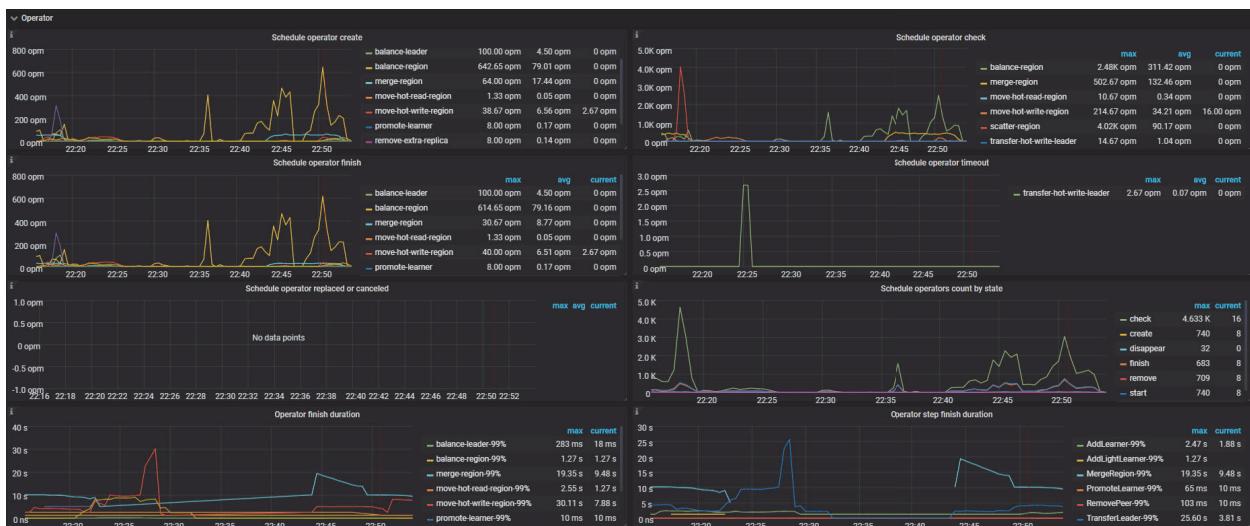


Figure 179: PD Dashboard - Operator metrics

#### 11.2.3.4 Statistics - Balance

- Store capacity: The capacity size per TiKV instance
- Store available: The available capacity size per TiKV instance
- Store used: The used capacity size per TiKV instance
- Size amplification: The size amplification ratio per TiKV instance, which is equal to (Store Region size)/(Store used capacity size)
- Size available ratio: The size availability ratio per TiKV instance, which is equal to (Store available capacity size)/(Store capacity size)
- Store leader score: The leader score per TiKV instance
- Store Region score: The Region score per TiKV instance
- Store leader size: The total leader size per TiKV instance
- Store Region size: The total Region size per TiKV instance
- Store leader count: The leader count per TiKV instance
- Store Region count: The Region count per TiKV instance



Figure 180: PD Dashboard - Balance metrics

#### 11.2.3.5 Statistics - hot write

- Hot Region's leader distribution: The total number of leader Regions that have become write hotspots on each TiKV instance
- Total written bytes on hot leader Regions: The total written bytes by leader Regions that have become write hotspots on each TiKV instance
- Hot write Region's peer distribution: The total number of peer Regions that have become write hotspots on each TiKV instance
- Total written bytes on hot peer Regions: The written bytes of all peer Regions that have become write hotspots on each TiKV instance
- Store Write rate bytes: The total written bytes on each TiKV instance
- Store Write rate keys: The total written keys on each TiKV instance
- Hot cache write entry number: The number of peers on each TiKV instance that are in the write hotspot statistics module
- Selector events: The event count of Selector in the hotspot scheduling module
- Direction of hotspot move leader: The direction of leader movement in the hotspot scheduling. The positive number means scheduling into the instance. The negative number means scheduling out of the instance
- Direction of hotspot move peer: The direction of peer movement in the hotspot scheduling. The positive number means scheduling into the instance. The negative number means scheduling out of the instance

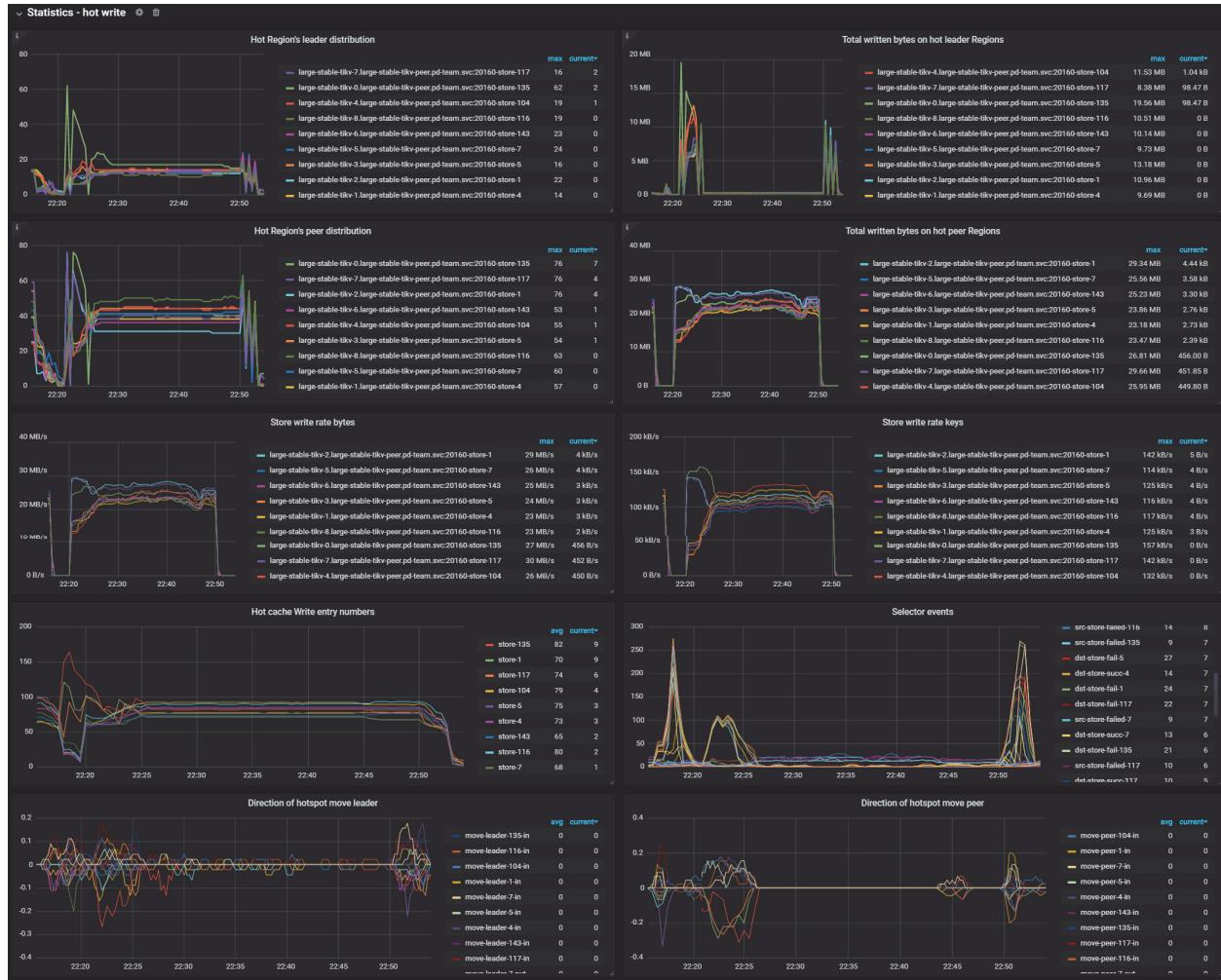


Figure 181: PD Dashboard - Hot write metrics

#### 11.2.3.6 Statistics - hot read

- Hot Region's peer distribution: The total number of peer Regions that have become read hotspots on each TiKV instance
- Total read bytes on hot peer Regions: The total read bytes of peers that have become read hotspots on each TiKV instance
- Store read rate bytes: The total read bytes of each TiKV instance
- Store read rate keys: The total read keys of each TiKV instance
- Hot cache read entry number: The number of peers that are in the read hotspot statistics module on each TiKV instance

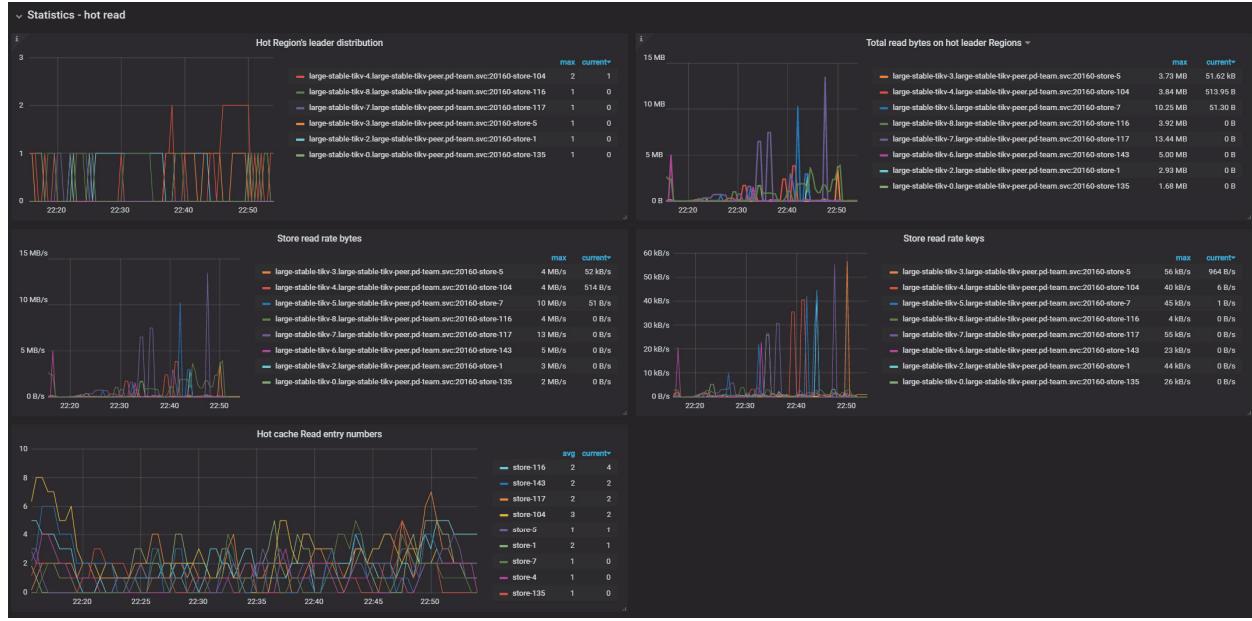


Figure 182: PD Dashboard - Hot read metrics

### 11.2.3.7 Scheduler

- Scheduler is running: The current running schedulers
- Balance leader movement: The leader movement details among TiKV instances
- Balance Region movement: The Region movement details among TiKV instances
- Balance leader event: The count of balance leader events
- Balance Region event: The count of balance Region events
- Balance leader scheduler: The inner status of balance leader scheduler
- Balance Region scheduler: The inner status of balance Region scheduler
- Replica checker: The replica checker's status
- Rule checker: The rule checker's status
- Region merge checker: The merge checker's status
- Filter target: The number of attempts that the store is selected as the scheduling target but failed to pass the filter
- Filter source: The number of attempts that the store is selected as the scheduling source but failed to pass the filter
- Balance Direction: The number of times that the Store is selected as the target or source of scheduling
- Store Limit: The flow control limitation of scheduling on the Store

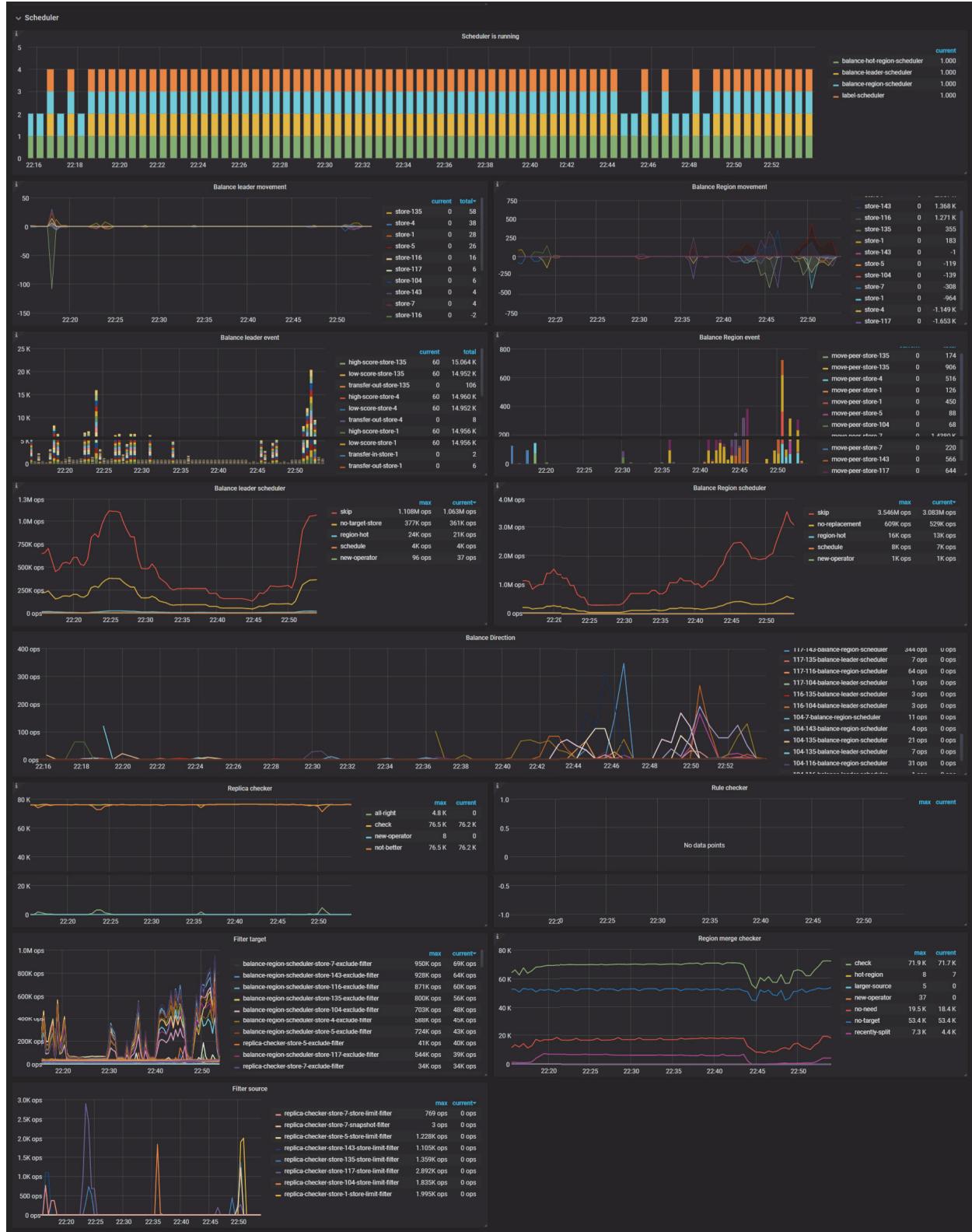


Figure 183: PD Dashboard - Scheduler metrics

### 11.2.3.8 gRPC

- Completed commands rate: The rate per command type at which gRPC commands are completed
- 99% Completed commands duration: The rate per command type at which gRPC commands are completed (P99)



Figure 184: PD Dashboard - gRPC metrics

### 11.2.3.9 etcd

- Handle transactions count: The rate at which etcd handles transactions
- 99% Handle transactions duration: The transaction handling rate (P99)
- 99% WAL fsync duration: The time consumed for writing WAL into the persistent storage. It is less than 1s (P99)
- 99% Peer round trip time seconds: The network latency for etcd (P99) | The value is less than 1s
- etcd disk WAL fsync rate: The rate of writing WAL into the persistent storage
- Raft term: The current term of Raft
- Raft committed index: The last committed index of Raft
- Raft applied index: The last applied index of Raft



Figure 185: PD Dashboard - etcd metrics

#### 11.2.3.10 TiDB

- PD Server TSO handle time and Client recv time: The duration between PD receiving the TSO request and the PD client getting the TSO response
- Handle requests count: The count of TiDB requests
- Handle requests duration: The time consumed for handling TiDB requests. It should be less than 100ms (P99)

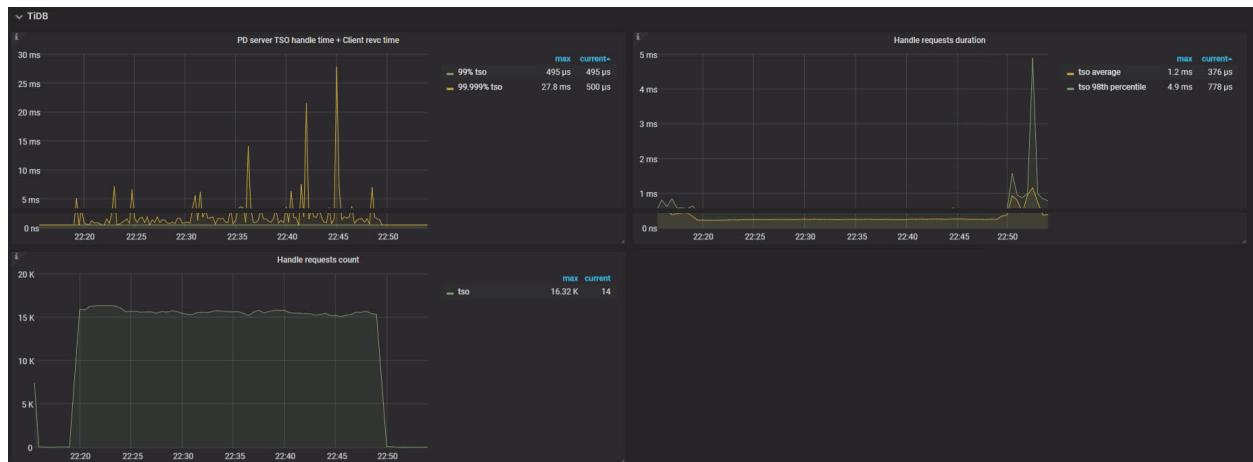


Figure 186: PD Dashboard - TiDB metrics

### 11.2.3.11 Heartbeat

- Heartbeat region event QPS: The QPS of handling heartbeat messages, including updating the cache and persisting data
- Region heartbeat report: The count of heartbeats reported to PD per instance
- Region heartbeat report error: The count of heartbeats with the `error` status
- Region heartbeat report active: The count of heartbeats with the `ok` status
- Region schedule push: The count of corresponding schedule commands sent from PD per TiKV instance
- 99% Region heartbeat latency: The heartbeat latency per TiKV instance (P99)



Figure 187: PD Dashboard - Heartbeat metrics

### 11.2.3.12 Region storage

- Syncer Index: The maximum index in the Region change history recorded by the leader
- history last index: The last index where the Region change history is synchronized successfully with the follower



Figure 188: PD Dashboard - Region storage

#### 11.2.4 Key Monitoring Metrics of TiKV

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus/-Grafana) is deployed at the same time. For more information, see [Overview of the Monitoring Framework](#).

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node\_exporter, and so on. A lot of metrics are there to help you diagnose.

You can get an overview of the component TiKV status from the **TiKV-Details** dashboard, where the key metrics are displayed. According to the [Performance Map](#), you can check whether the status of the cluster is as expected.

This document provides a detailed description of these key metrics on the **TiKV-Details** dashboard.

##### 11.2.4.1 Cluster

- Store size: The storage size per TiKV instance
- Available size: The available capacity per TiKV instance
- Capacity size: The capacity size per TiKV instance
- CPU: The CPU utilization per TiKV instance
- Memory: The memory usage per TiKV instance
- IO utilization: The I/O utilization per TiKV instance
- MBps: The total bytes of read and write in each TiKV instance
- QPS: The QPS per command in each TiKV instance
- Errps: The rate of gRPC message failures
- leader: The number of leaders per TiKV instance
- Region: The number of Regions per TiKV instance
- Uptime: The runtime of TiKV since last restart

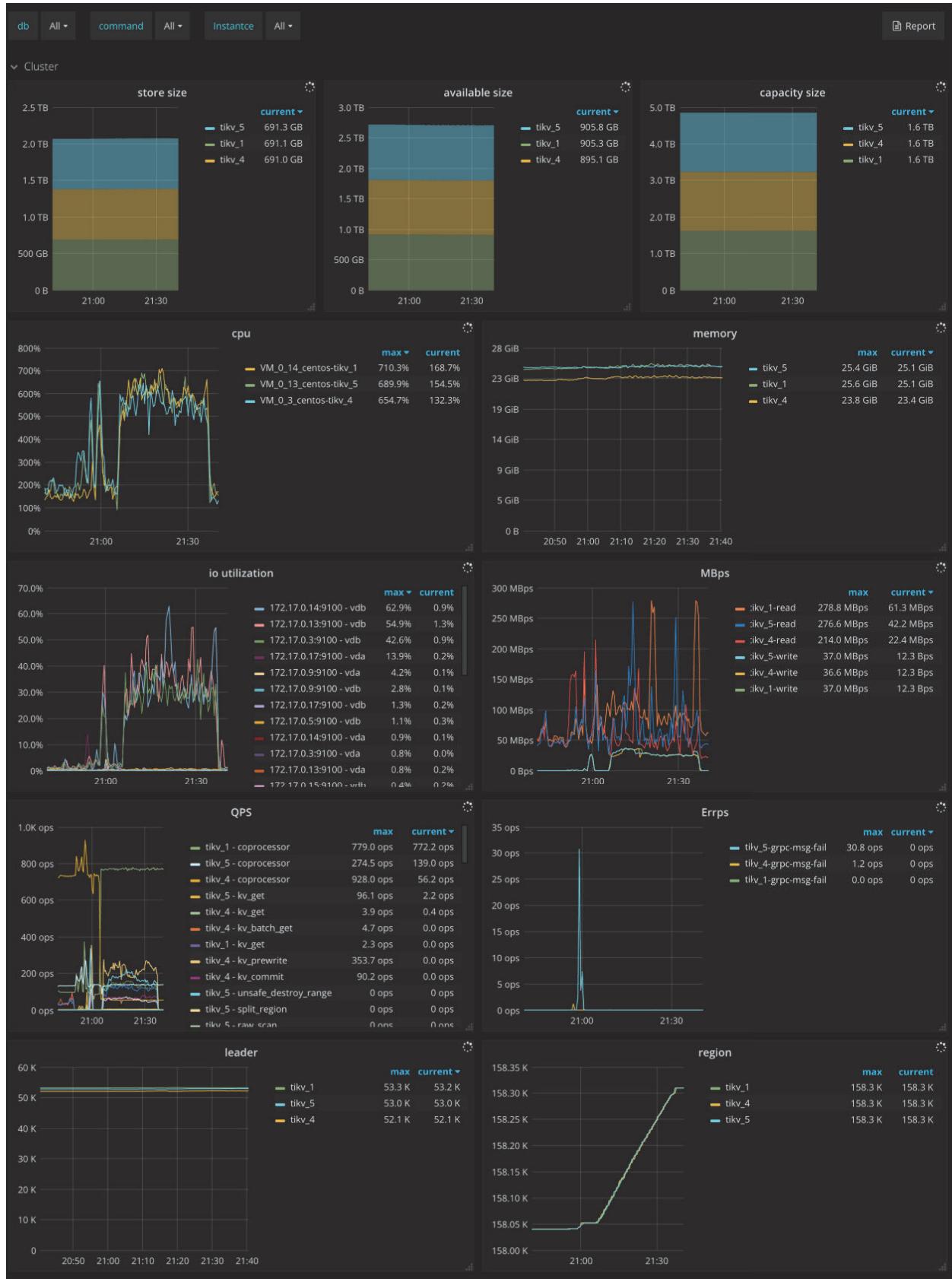


Figure 189: TiKV Dashboard - Cluster metrics  
1342

#### 11.2.4.2 Errors

- Critical error: The number of critical errors
- Server is busy: Indicates occurrences of events that make the TiKV instance unavailable temporarily, such as Write Stall, Channel Full, and so on. It should be 0 in normal case.
- Server report failures: The number of error messages reported by server. It should be 0 in normal case.
- Raftstore error: The number of Raftstore errors per type on each TiKV instance
- Scheduler error: The number of scheduler errors per type on each TiKV instance
- Coprocessor error: The number of coprocessor errors per type on each TiKV instance
- gRPC message error: The number of gRPC message errors per type on each TiKV instance
- Leader drop: The count of dropped leaders per TiKV instance
- Leader missing: The count of missing leaders per TiKV instance

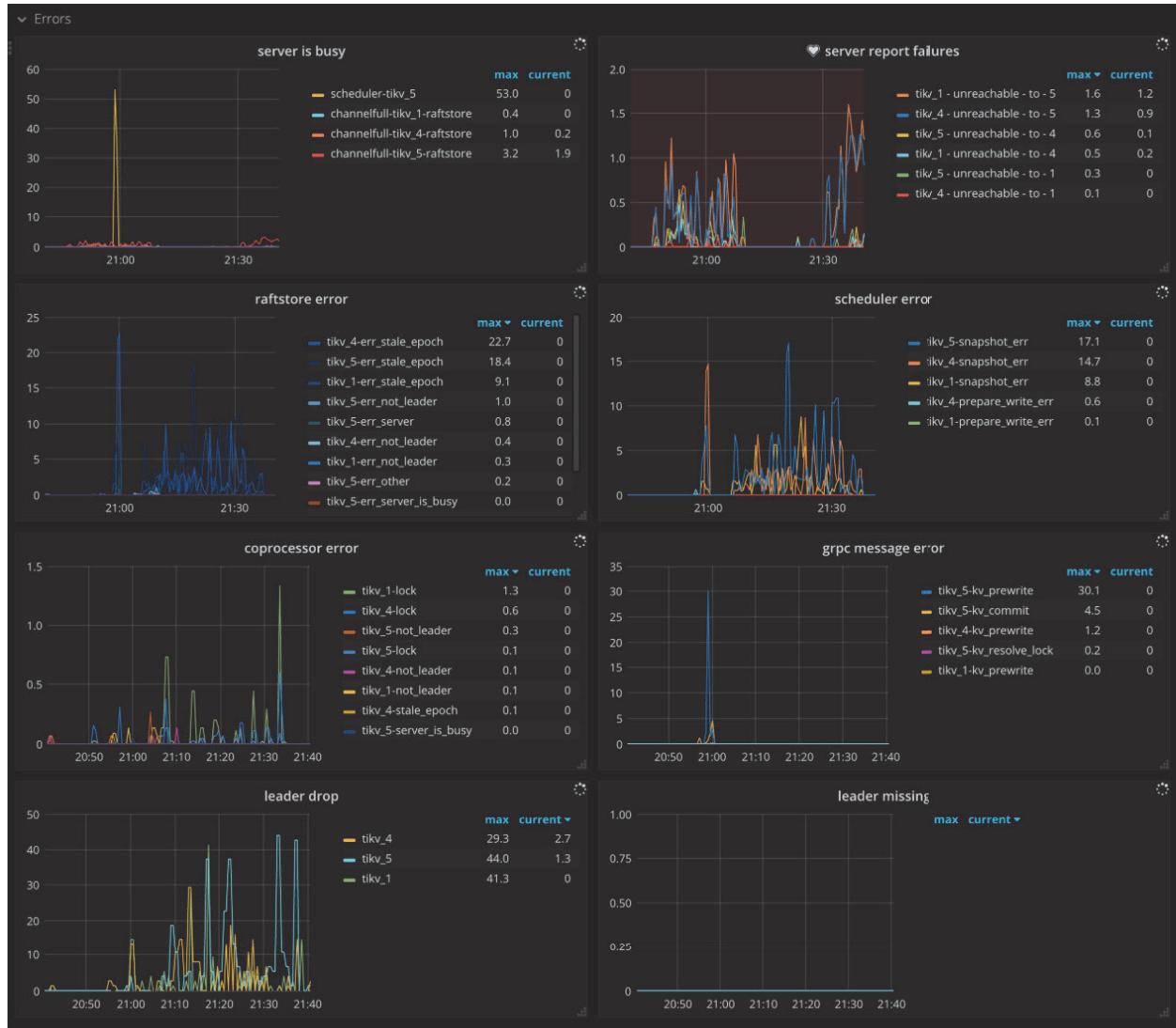


Figure 190: TiKV Dashboard - Errors metrics

#### 11.2.4.3 Server

- CF size: The size of each column family
- Store size: The storage size per TiKV instance
- Channel full: The number of Channel Full errors per TiKV instance. It should be 0 in normal case.
- Active written leaders: The number of leaders being written on each TiKV instance
- Approximate Region size: The approximate Region size
- Approximate Region size Histogram: The histogram of each approximate Region size
- Region average written keys: The average number of written keys to Regions per TiKV instance
- Region average written bytes: The average written bytes to Regions per TiKV instance

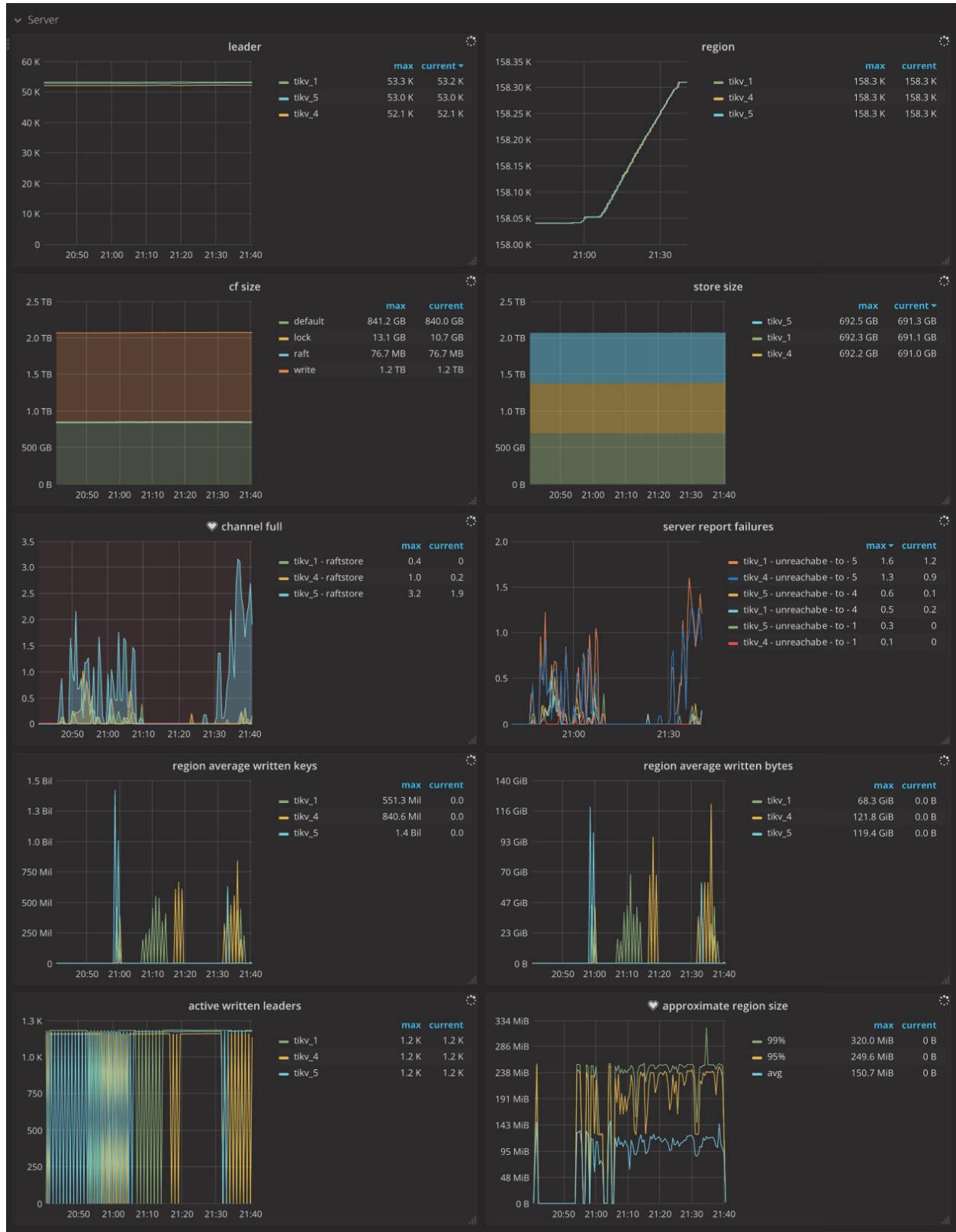


Figure 191: TiKV Dashboard - Server metrics

#### 11.2.4.4 gRPC

- gRPC message count: The rate of gRPC messages per type
- gRPC message failed: The rate of failed gRPC messages
- 99% gRPC message duration: The gRPC message duration per message type (P99)
- Average gRPC message duration: The average execution time of gRPC messages
- gRPC batch size: The batch size of gRPC messages between TiDB and TiKV
- Raft message batch size: The batch size of Raft messages between TiKV instances

#### 11.2.4.5 Thread CPU

- Raft store CPU: The CPU utilization of the `raftstore` thread. The CPU utilization should be less than  $80\% * \text{raftstore.store-pool-size}$  in normal case.
- Async apply CPU: The CPU utilization of the `async apply` thread. The CPU utilization should be less than  $90\% * \text{raftstore.apply-pool-size}$  in normal cases.
- Scheduler worker CPU: The CPU utilization of the `scheduler worker` thread. The CPU utilization should be less than  $90\% * \text{storage.scheduler-worker-pool-size}$  in normal cases.
- gRPC poll CPU: The CPU utilization of the `gRPC` thread. The CPU utilization should be less than  $80\% * \text{server.grpc-concurrency}$  in normal cases.
- Unified read pool CPU: The CPU utilization of the `unified read pool` thread
- Storage ReadPool CPU: The CPU utilization of the `storage read pool` thread
- Coprocessor CPU: The CPU utilization of the `coprocessor` thread
- RocksDB CPU: The CPU utilization of the RocksDB thread
- GC worker CPU: The CPU utilization of the `GC worker` thread
- BackGround worker CPU: The CPU utilization of the `background worker` thread

#### 11.2.4.6 PD

- PD requests: The rate at which TiKV sends to PD
- PD request duration (average): The average duration of processing requests that TiKV sends to PD
- PD heartbeats: The rate at which heartbeat messages are sent from TiKV to PD
- PD validate peers: The rate at which messages are sent from TiKV to PD to validate TiKV peers

#### 11.2.4.7 Raft IO

- Apply log duration: The time consumed for Raft to apply logs
- Apply log duration per server: The time consumed for Raft to apply logs per TiKV instance
- Append log duration: The time consumed for Raft to append logs

- Append log duration per server: The time consumed for Raft to append logs per TiKV instance
- Commit log duration: The time consumed by Raft to commit logs
- Commit log duration per server: The time consumed by Raft to commit logs per TiKV instance

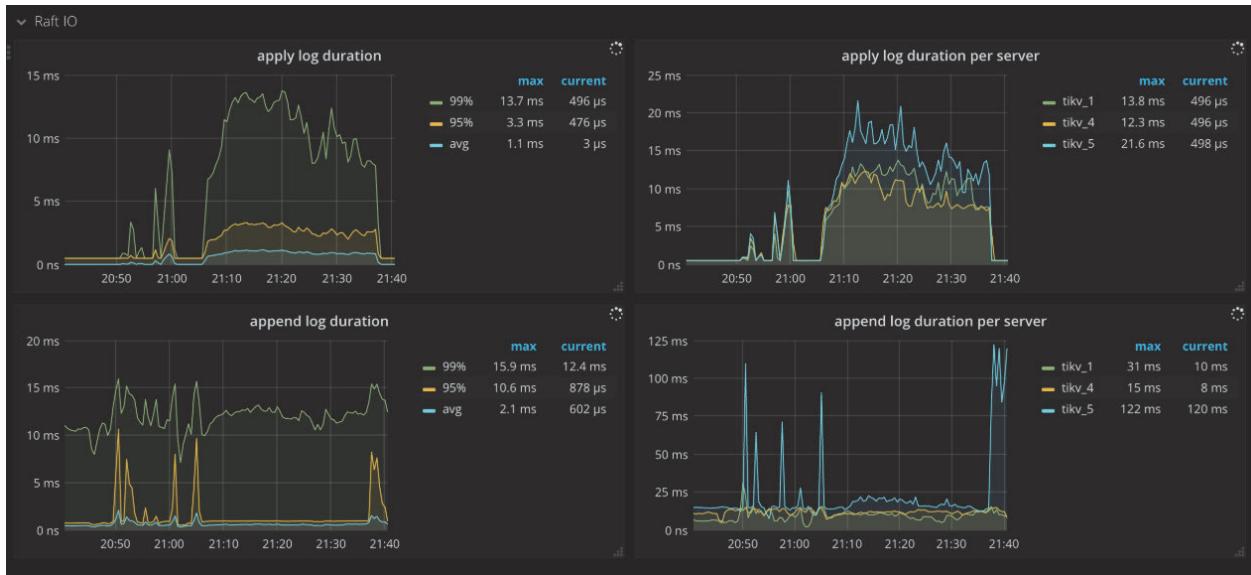


Figure 192: TiKV Dashboard - Raft IO metrics

#### 11.2.4.8 Raft process

- Ready handled: The number of handled ready operations per type per second
  - count: The number of handled ready operations per second
  - has\_ready\_region: The number of Regions that have ready per second
  - pending\_region: The operations per second of the Regions being checked for whether it has ready. This metric is deprecated since v3.0.0
  - message: The number of messages that the ready operations per second contain
  - append: The number of Raft log entries that the ready operations per second contain
  - commit: The number of committed Raft log entries that the ready operations per second contain
  - snapshot: The number of snapshots that the ready operations per second contains
- 0.99 Duration of Raft store events: The time consumed by Raftstore events (P99)
- Process ready duration: The time consumed for processes to be ready in Raft
- Process ready duration per server: The time consumed for peer processes to be ready in Raft per TiKV instance. It should be less than 2 seconds (P99.99).

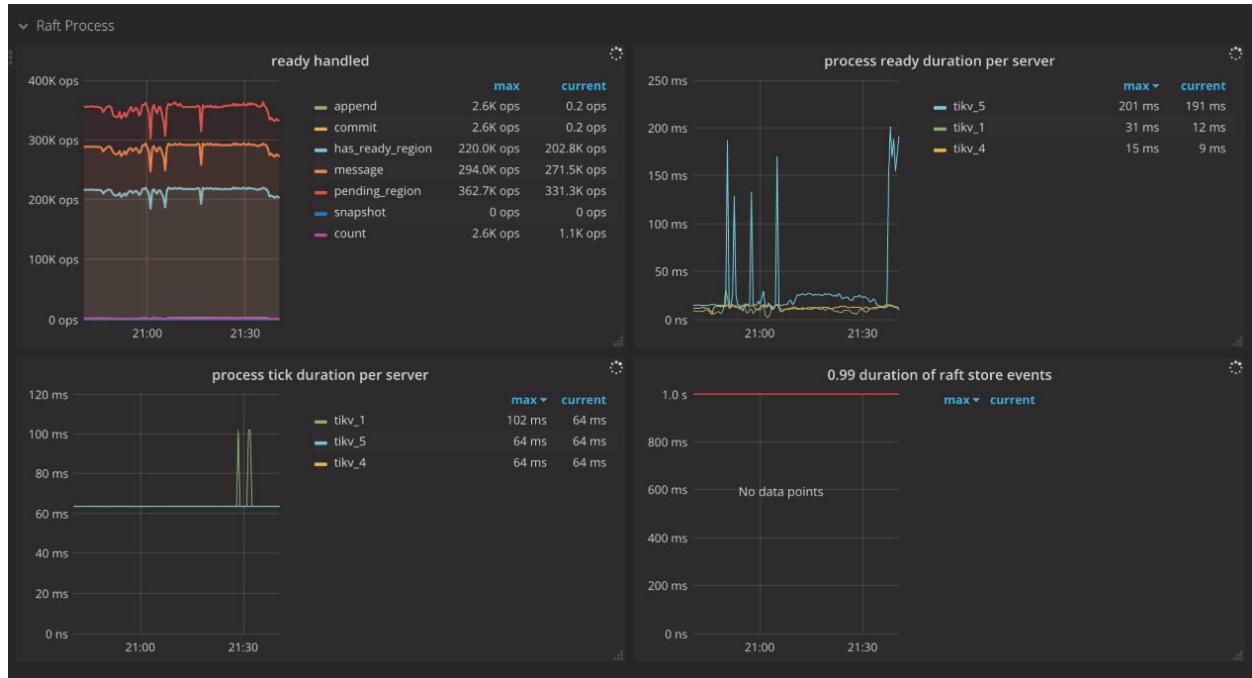


Figure 193: TiKV Dashboard - Raft process metrics

#### 11.2.4.9 Raft message

- Sent messages per server: The number of Raft messages sent by each TiKV instance per second
- Flush messages per server: The number of Raft messages flushed by the Raft client in each TiKV instance per second
- Receive messages per server: The number of Raft messages received by each TiKV instance per second
- Messages: The number of Raft messages sent per type per second
- Vote: The number of Vote messages sent in Raft per second
- Raft dropped messages: The number of dropped Raft messages per type per second

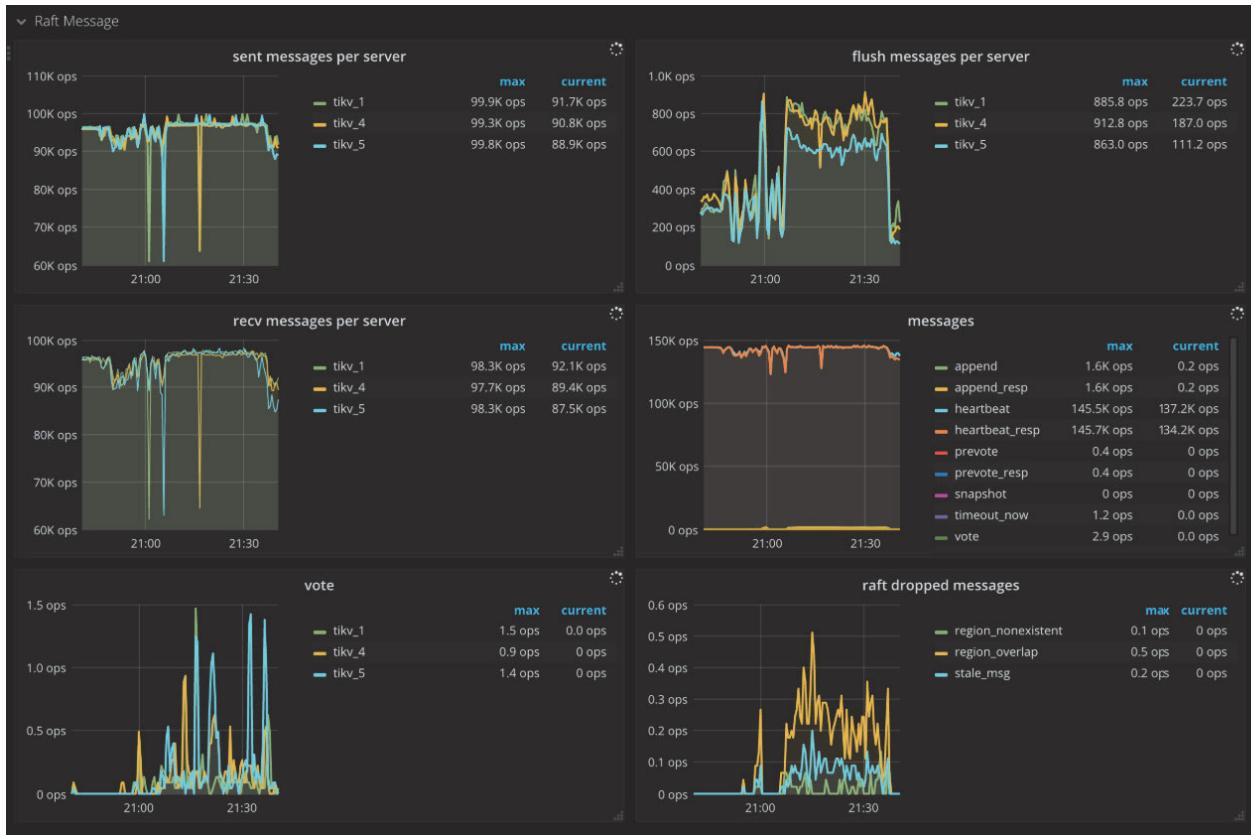


Figure 194: TiKV Dashboard - Raft message metrics

#### 11.2.4.10 Raft propose

- Raft apply proposals per ready: The histogram of the number of proposals that each ready operation contains in a batch while applying proposal.
- Raft read/write proposals: The number of proposals per type per second
- Raft read proposals per server: The number of read proposals made by each TiKV instance per second
- Raft write proposals per server: The number of write proposals made by each TiKV instance per second
- Propose wait duration: The histogram of waiting time of each proposal
- Propose wait duration per server: The histogram of waiting time of each proposal per TiKV instance
- Apply wait duration: The histogram of apply time of each proposal
- Apply wait duration per server: The histogram of apply time of each proposal per TiKV instance
- Raft log speed: The average rate at which peers propose logs

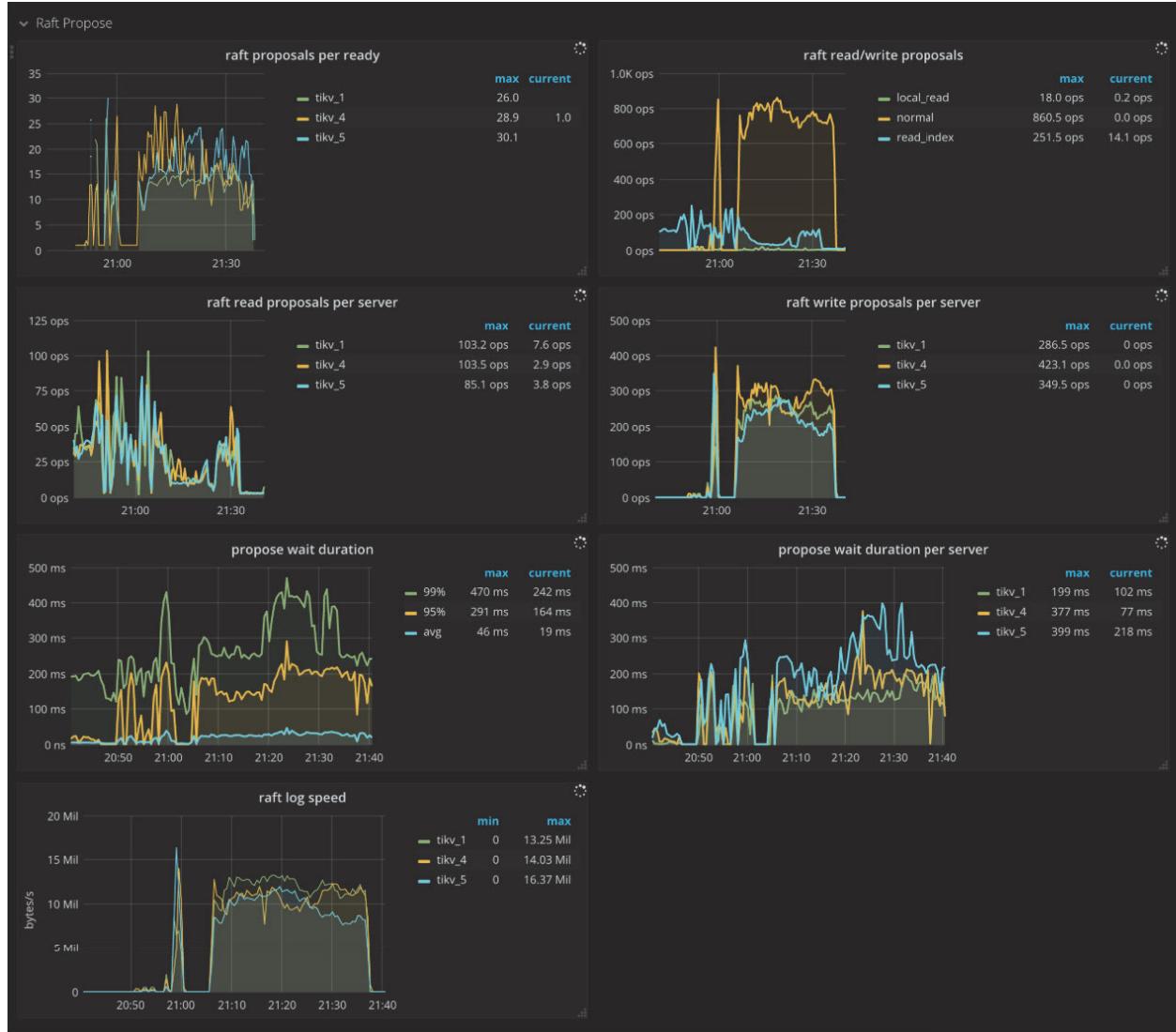


Figure 195: TiKV Dashboard - Raft propose metrics

#### 11.2.4.11 Raft admin

- Admin proposals: The number of admin proposals per second
- Admin apply: The number of processed apply commands per second
- Check split: The number of Raftstore split check commands per second
- 99.99% Check split duration: The time consumed when running split check commands (P99.99)

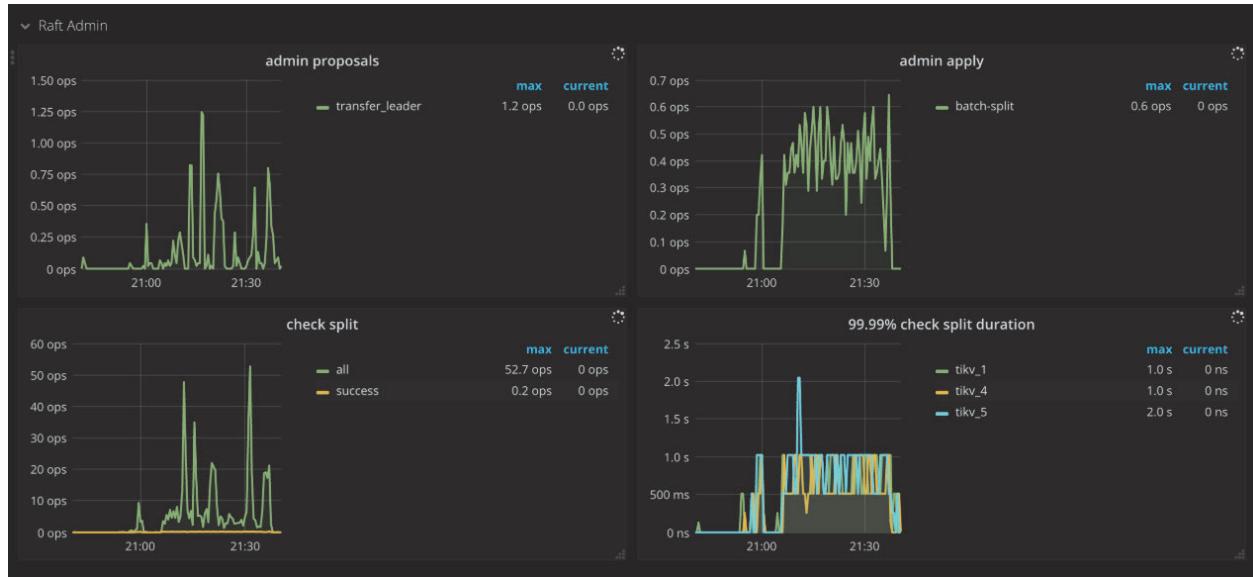


Figure 196: TiKV Dashboard - Raft admin metrics

#### 11.2.4.12 Local reader

- Local reader requests: The number of total requests and the number of rejections from the local read thread

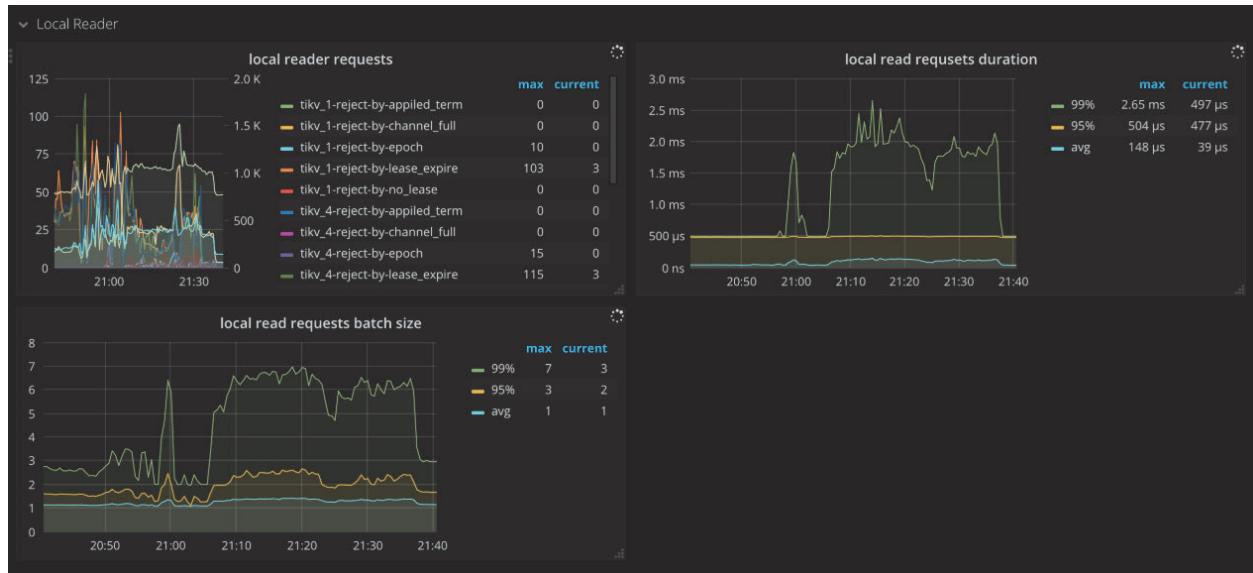


Figure 197: TiKV Dashboard - Local reader metrics

#### 11.2.4.13 Unified Read Pool

- Time used by level: The time consumed for each level in the unified read pool. Level 0 means small queries.
- Level 0 chance: The proportion of level 0 tasks in unified read pool
- Running tasks: The number of tasks running concurrently in the unified read pool

#### 11.2.4.14 Storage

- Storage command total: The number of received command by type per second
- Storage async request error: The number of engine asynchronous request errors per second
- Storage async snapshot duration: The time consumed by processing asynchronous snapshot requests. It should be less than 1s in .99.
- Storage async write duration: The time consumed by processing asynchronous write requests. It should be less than 1s in .99.

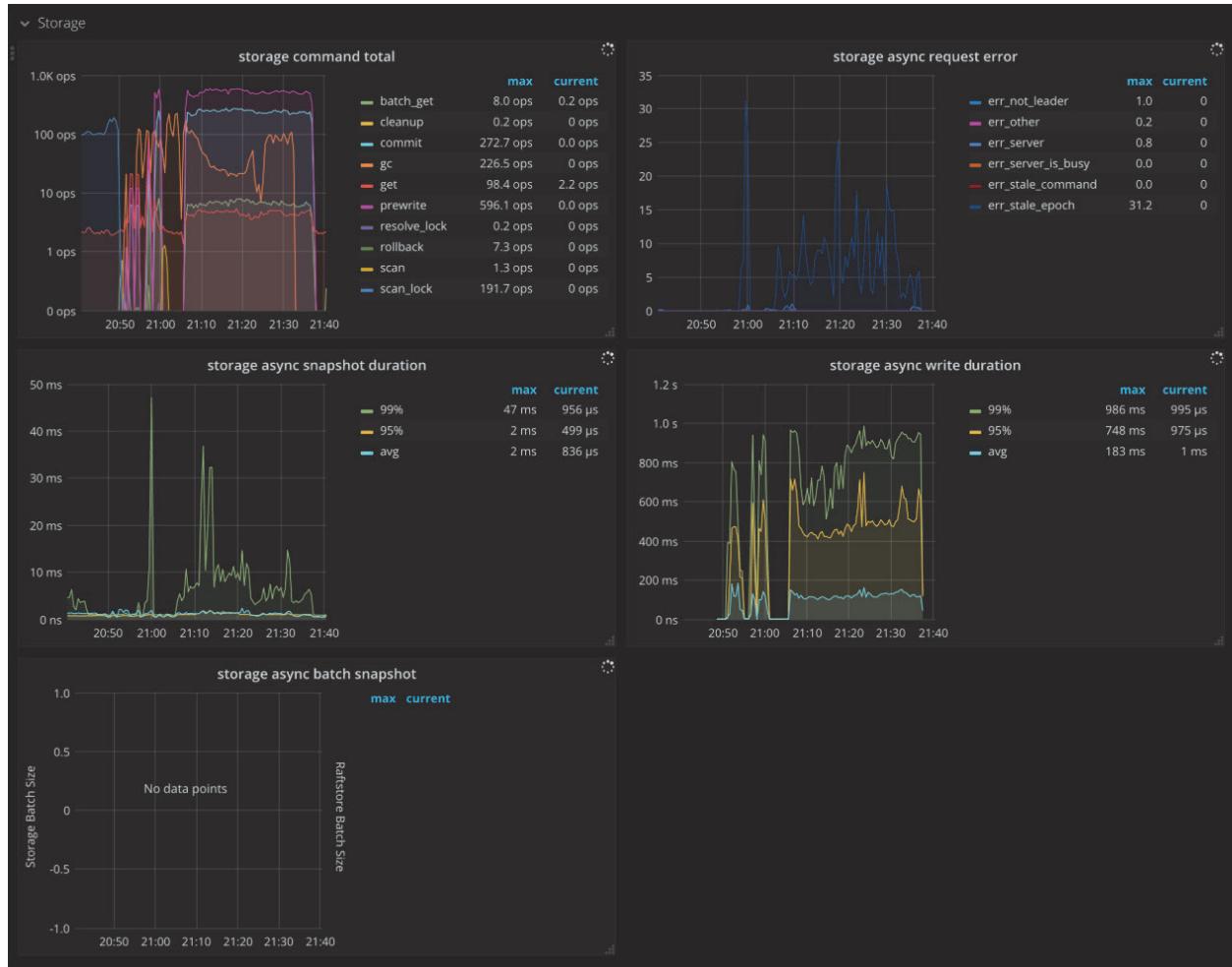


Figure 198: TiKV Dashboard - Storage metrics

#### 11.2.4.15 Scheduler

- Scheduler stage total: The number of commands at each stage per second. There should not be a lot of errors in a short time.
- Scheduler writing bytes: The total written bytes by commands processed on each TiKV instance
- Scheduler priority commands: The count of different priority commands per second
- Scheduler pending commands: The count of pending commands per TiKV instance per second

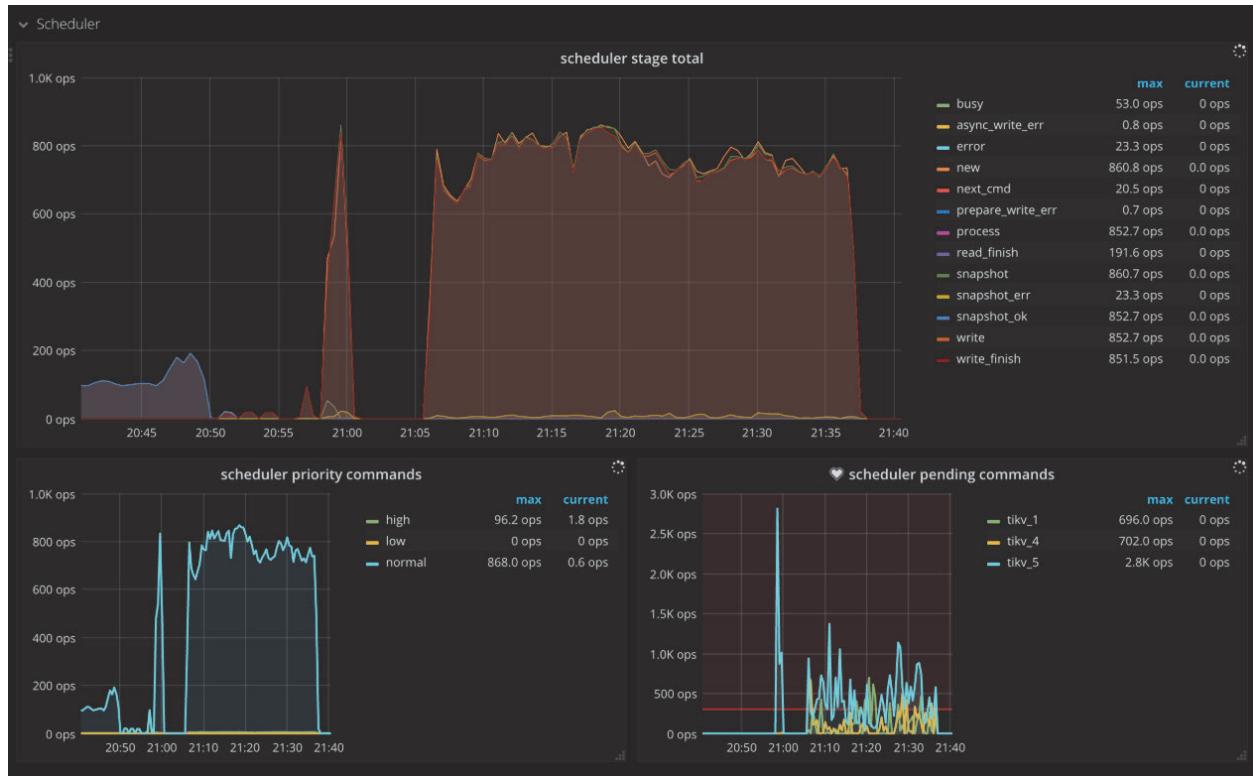


Figure 199: TiKV Dashboard - Scheduler metrics

#### 11.2.4.16 Scheduler - commit

- Scheduler stage total: The number of commands at each stage per second when executing the commit command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the commit command. It should be less than 1s.
- Scheduler latch wait duration: The waiting time caused by latch when executing the commit command. It should be less than 1s.
- Scheduler keys read: The count of keys read by a commit command
- Scheduler keys written: The count of keys written by a commit command
- Scheduler scan details: The keys scan details of each CF when executing the commit command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the commit command
- Scheduler scan details write: The keys scan details of write CF when executing the commit command
- Scheduler scan details [default]: The keys scan details of default CF when executing the commit command



Figure 200: TiKV Dashboard - Scheduler commit metrics

#### 11.2.4.17 Scheduler - pessimistic\_rollback

- Scheduler stage total: The number of commands at each stage per second when executing the `pessimistic_rollback` command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the `pessimistic_rollback` command. It should be less than 1s.

- Scheduler latch wait duration: The waiting time caused by latch when executing the `pessimistic_rollback` command. It should be less than `1s`.
- Scheduler keys read: The count of keys read by a `pessimistic_rollback` command
- Scheduler keys written: The count of keys written by a `pessimistic_rollback` command
- Scheduler scan details: The keys scan details of each CF when executing the `pessimistic_rollback` command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the `pessimistic_rollback` command
- Scheduler scan details `write`: The keys scan details of write CF when executing the `pessimistic_rollback` command
- Scheduler scan details [default]: The keys scan details of default CF when executing the `pessimistic_rollback` command

#### 11.2.4.18 Scheduler - prewrite

- Scheduler stage total: The number of commands at each stage per second when executing the prewrite command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the prewrite command. It should be less than `1s`.
- Scheduler latch wait duration: The waiting time caused by latch when executing the prewrite command. It should be less than `1s`.
- Scheduler keys read: The count of keys read by a prewrite command
- Scheduler keys written: The count of keys written by a prewrite command
- Scheduler scan details: The keys scan details of each CF when executing the prewrite command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the prewrite command
- Scheduler scan details `write`: The keys scan details of write CF when executing the prewrite command
- Scheduler scan details [default]: The keys scan details of default CF when executing the prewrite command

#### 11.2.4.19 Scheduler - rollback

- Scheduler stage total: The number of commands at each stage per second when executing the rollback command. There should not be a lot of errors in a short time.
- Scheduler command duration: The time consumed when executing the rollback command. It should be less than `1s`.
- Scheduler latch wait duration: The waiting time caused by latch when executing the rollback command. It should be less than `1s`.
- Scheduler keys read: The count of keys read by a rollback command
- Scheduler keys written: The count of keys written by a rollback command

- Scheduler scan details: The keys scan details of each CF when executing the rollback command.
- Scheduler scan details [lock]: The keys scan details of lock CF when executing the rollback command
- Scheduler scan details **write**: The keys scan details of write CF when executing the rollback command
- Scheduler scan details [default]: The keys scan details of default CF when executing the rollback command

#### 11.2.4.20 GC

- GC tasks: The count of GC tasks processed by gc\_worker
- GC tasks Duration: The time consumed when executing GC tasks
- TiDB GC seconds: The GC duration
- TiDB GC worker actions: The count of TiDB GC worker actions
- ResolveLocks Progress: The progress of the first phase of GC (Resolve Locks)
- TiKV Auto GC Progress: The progress of the second phase of GC
- GC speed: The number of keys deleted by GC per second
- TiKV Auto GC SafePoint: The value of TiKV GC safe point. The safe point is the current GC timestamp
- GC lifetime: The lifetime of TiDB GC
- GC interval: The interval of TiDB GC
- GC in Compaction Filter: The count of filtered versions in the compaction filter of write CF.

#### 11.2.4.21 Snapshot

- Rate snapshot message: The rate at which Raft snapshot messages are sent
- 99% Handle snapshot duration: The time consumed to handle snapshots (P99)
- Snapshot state count: The number of snapshots per state
- 99.99% Snapshot size: The snapshot size (P99.99)
- 99.99% Snapshot KV count: The number of KV within a snapshot (P99.99)

#### 11.2.4.22 Task

- Worker handled tasks: The number of tasks handled by worker per second
- Worker pending tasks: Current number of pending and running tasks of worker per second. It should be less than 1000 in normal case.
- FuturePool handled tasks: The number of tasks handled by future pool per second
- FuturePool pending tasks: Current number of pending and running tasks of future pool per second

#### 11.2.4.23 Coprocessor Overview

- Request duration: The total duration from the time of receiving the coprocessor request to the time of finishing processing the request
- Total Requests: The number of requests by type per second
- Handle duration: The histogram of time spent actually processing coprocessor requests per minute
- Total Request Errors: The number of request errors of Coprocessor per second. There should not be a lot of errors in a short time.
- Total KV Cursor Operations: The total number of the KV cursor operations by type per second, such as `select`, `index`, `analyze_table`, `analyze_index`, `checksum_table ↞`, `checksum_index`, and so on.
- KV Cursor Operations: The histogram of KV cursor operations by type per second
- Total RocksDB Perf Statistics: The statistics of RocksDB performance
- Total Response Size: The total size of coprocessor response

#### 11.2.4.24 Coprocessor Detail

- Handle duration: The histogram of time spent actually processing coprocessor requests per minute
- 95% Handle duration by store: The time consumed to handle coprocessor requests per TiKV instance per second (P95)
- Wait duration: The time consumed when coprocessor requests are waiting to be handled. It should be less than 10s (P99.99).
- 95% Wait duration by store: The time consumed when coprocessor requests are waiting to be handled per TiKV instance per second (P95)
- Total DAG Requests: The total number of DAG requests per second
- Total DAG Executors: The total number of DAG executors per second
- Total Ops Details (Table Scan): The number of RocksDB internal operations per second when executing select scan in coprocessor
- Total Ops Details (Index Scan): The number of RocksDB internal operations per second when executing index scan in coprocessor
- Total Ops Details by CF (Table Scan): The number of RocksDB internal operations for each CF per second when executing select scan in coprocessor
- Total Ops Details by CF (Index Scan): The number of RocksDB internal operations for each CF per second when executing index scan in coprocessor

#### 11.2.4.25 Threads

- Threads state: The state of TiKV threads
- Threads IO: The I/O traffic of each TiKV thread
- Thread Voluntary Context Switches: The number of TiKV threads voluntary context switches
- Thread Nonvoluntary Context Switches: The number of TiKV threads nonvoluntary context switches

#### 11.2.4.26 RocksDB - kv/raft

- Get operations: The count of get operations per second
- Get duration: The time consumed when executing get operations
- Seek operations: The count of seek operations per second
- Seek duration: The time consumed when executing seek operations
- Write operations: The count of write operations per second
- Write duration: The time consumed when executing write operations
- WAL sync operations: The count of WAL sync operations per second
- Write WAL duration: The time consumed for writing WAL
- WAL sync duration: The time consumed when executing WAL sync operations
- Compaction operations: The count of compaction and flush operations per second
- Compaction duration: The time consumed when executing the compaction and flush operations
- SST read duration: The time consumed when reading SST files
- Write stall duration: Write stall duration. It should be 0 in normal case.
- Memtable size: The memtable size of each column family
- Memtable hit: The hit rate of memtable
- Block cache size: The block cache size. Broken down by column family if shared block cache is disabled.
- Block cache hit: The hit rate of block cache
- Block cache flow: The flow rate of block cache operations per type
- Block cache operations: The count of block cache operations per type
- Keys flow: The flow rate of operations on keys per type
- Total keys: The count of keys in each column family
- Read flow: The flow rate of read operations per type
- Bytes / Read: The bytes per read operation
- Write flow: The flow rate of write operations per type
- Bytes / Write: The bytes per write operation
- Compaction flow: The flow rate of compaction operations per type
- Compaction pending bytes: The pending bytes to be compacted
- Read amplification: The read amplification per TiKV instance
- Compression ratio: The compression ratio of each level
- Number of snapshots: The number of snapshots per TiKV instance
- Oldest snapshots duration: The time that the oldest unreleased snapshot survivals
- Number files at each level: The number of SST files for different column families in each level
- Ingest SST duration seconds: The time consumed to ingest SST files
- Stall conditions changed of each CF: Stall conditions changed of each column family

#### 11.2.4.27 Titan - All

- Blob file count: The number of Titan blob files
- Blob file size: The total size of Titan blob file

- Live blob size: The total size of valid blob record
- Blob cache hit: The hit rate of Titan block cache
- Iter touched blob file count: The number of blob file involved in a single iterator
- Blob file discardable ratio distribution: The ratio distribution of blob record failure of blob files
- Blob key size: The size of Titan blob keys
- Blob value size: The size of Titan blob values
- Blob get operations: The count of get operations in Titan blob
- Blob get duration: The time consumed when executing get operations in Titan blob
- Blob iter operations: The time consumed when executing iter operations in Titan blob
- Blob seek duration: The time consumed when executing seek operations in Titan blob
- Blob next duration: The time consumed when executing next operations in Titan blob
- Blob prev duration: The time consumed when executing prev operations in Titan blob
- Blob keys flow: The flow rate of operations on Titan blob keys
- Blob bytes flow: The flow rate of bytes on Titan blob keys
- Blob file read duration: The time consumed when reading Titan blob file
- Blob file write duration: The time consumed when writing Titan blob file
- Blob file sync operations: The count of blob file sync operations
- Blob file sync duration: The time consumed when synchronizing blob file
- Blob GC action: The count of Titan GC actions
- Blob GC duration: The Titan GC duration
- Blob GC keys flow: The flow rate of keys read and written by Titan GC
- Blob GC bytes flow: The flow rate of bytes read and written by Titan GC
- Blob GC input file size: The size of Titan GC input file
- Blob GC output file size: The size of Titan GC output file
- Blob GC file count: The count of blob files involved in Titan GC

#### 11.2.4.28 Lock manager

- Thread CPU: The CPU utilization of the lock manager thread
- Handled tasks: The number of tasks handled by lock manager
- Waiter lifetime duration: The waiting time of the transaction for the lock to be released
- Wait table: The status information of wait table, including the number of locks and the number of transactions waiting for the lock
- Deadlock detect duration: The time consumed for detecting deadlock
- Detect error: The number of errors encountered when detecting deadlock, including the number of deadlocks
- Deadlock detector leader: The information of the node where the deadlock detector leader is located

#### 11.2.4.29 Memory

- Allocator Stats: The statistics of the memory allocator

#### 11.2.4.30 Backup

- Backup CPU: The CPU utilization of the backup thread
- Range Size: The histogram of backup range size
- Backup Duration: The time consumed for backup
- Backup Flow: The total bytes of backup
- Disk Throughput: The disk throughput per instance
- Backup Range Duration: The time consumed for backing up a range
- Backup Errors: The number of errors encountered during a backup

#### 11.2.4.31 Encryption

- Encryption data keys: The total number of encrypted data keys
- Encrypted files: The number of encrypted files
- Encryption initialized: Shows whether encryption is enabled. 1 means enabled.
- Encryption meta files size: The size of the encryption meta file
- Encrypt/decrypt data nanos: The histogram of duration on encrypting/decrypting data each time
- Read/write encryption meta duration: The time consumed for reading/writing encryption meta files

#### 11.2.4.32 Explanation of Common Parameters

##### 11.2.4.32.1 gRPC Message Type

###### 1. Transactional API:

- kv\_get: The command of getting the latest version of data specified by ts
- kv\_scan: The command of scanning a range of data
- kv\_prewrite: The command of prewriting the data to be committed at first phase of 2PC
- kv\_pessimistic\_lock: The command of adding a pessimistic lock to the key to prevent other transaction from modifying this key
- kv\_pessimistic\_rollback: The command of deleting the pessimistic lock on the key
- kv\_txn\_heart\_beat: The command of updating lock\_ttl for pessimistic transactions or large transactions to prevent them from rolling back
- kv\_check\_txn\_status: The command of checking the status of the transaction
- kv\_commit: The command of committing the data written by the prewrite command
- kv\_cleanup: The command of rolling back a transaction, which is deprecated in v4.0

- `kv_batch_get`: The command of getting the value of batch key at once, similar to `kv_get`
- `kv_batch_rollback`: The command of batch rollback of multiple prewrite transactions
- `kv_scan_lock`: The command of scanning all locks with a version number before `max_version` to clean up expired transactions
- `kv_resolve_lock`: The command of committing or rollback the transaction lock, according to the transaction status.
- `kv_gc`: The command of GC
- `kv_delete_range`: The command of deleting a range of data from TiKV

## 2. Raw API:

- `raw_get`: The command of getting the value of key
- `raw_batch_get`: The command of getting the value of batch keys
- `raw_scan`: The command of scanning a range of data
- `raw_batch_scan`: The command of scanning multiple consecutive data range
- `raw_put`: The command of writing a key/value pair
- `raw_batch_put`: The command of writing a batch of key/value pairs
- `raw_delete`: The command of deleting a key/value pair
- `raw_batch_delete`: The command of a batch of key/value pairs
- `raw_delete_range`: The command of deleting a range of data

### 11.2.5 Monitor the TiFlash Cluster

This document describes the monitoring items of TiFlash.

If you use TiUP to deploy the TiDB cluster, the monitoring system (Prometheus & Grafana) is deployed at the same time. For more information, see [Overview of the Monitoring Framework](#).

The Grafana dashboard is divided into a series of sub dashboards which include Overview, PD, TiDB, TiKV, Node\_exporter, and so on. A lot of metrics are there to help you diagnose.

TiFlash has three dashboard panels: **TiFlash-Summary**, **TiFlash-Proxy-Summary**, and **TiFlash-Proxy-Details**. The metrics on these panels indicate the current status of TiFlash. The **TiFlash-Proxy-Summary** and **TiFlash-Proxy-Details** panels mainly show the information of the Raft layer and the metrics are detailed in [Key Monitoring Metrics of TiKV](#).

#### Note:

It is recommended that you use TiDB v4.0.5 or later versions for improved monitor on TiFlash.

The following sections introduce the default monitoring information of **TiFlash-Summary**.

#### 11.2.5.1 Server

- Store size: The storage size used by each TiFlash instance.
- Available size: The storage size available for each TiFlash instance.
- Capacity size: The storage capacity for each TiFlash instance.
- Uptime: The runtime of TiFlash since last restart.
- Memory: The memory usage per TiFlash instance.
- CPU Usage: The CPU utilization per TiFlash instance.
- FSync OPS: The number of fsync operations per TiFlash instance per second.
- File Open OPS: The number of open operations per TiFlash instance per second.
- Opened File Count: The number of file descriptors currently opened by each TiFlash instance.

**Note:**

Store size, FSync OPS, File Open OPS, and Opened File Count currently only cover the monitoring information of the TiFlash storage layer and do not cover that in TiFlash-Proxy.

#### 11.2.5.2 Coprocessor

- Request QPS: The number of coprocessor requests received by all TiFlash instances. `batch` is the number of batch requests. `batch_cop` is the number of coprocessor requests in the batch requests. `cop` is the number of coprocessor requests that are sent directly via the coprocessor interface. `cop_dag` is the number of dag requests in all coprocessor requests. `super_batch` is the number of requests to enable the Super Batch feature.
- Executor QPS: The number of each type of dag executors in the requests received by all TiFlash instances. `table_scan` is the table scan executor. `selection` is the selection executor. `aggregation` is the aggregation executor. `top_n` is the TopN executor. `limit` is the limit executor.
- Request Duration: The total duration of all TiFlash instances processing coprocessor requests. The total duration is from the time that the coprocessor request is received to the time that the response to the request is completed.
- Error QPS: The number of errors of all TiFlash instances processing coprocessor requests. `meet_lock` means that the read data is locked. `region_not_found` means that the Region does not exist. `epoch_not_match` means the read Region epoch is inconsistent with the local epoch. `kv_client_error` means that the communication

with TiKV returns an error. `internal_error` is the internal system error of TiFlash. `other` is other types of errors.

- Request Handle Duration: The duration of all TiFlash instances processing coprocessor requests. The processing time is from starting to execute the coprocessor request to completing the execution.
- Response Bytes/Seconds: The total bytes of the response from all TiFlash instances.
- Cop task memory usage: The total memory usage of all TiFlash instances processing coprocessor requests.
- Handling Request Number: The total number of all TiFlash instances processing co-processor requests. The classification of the requests is the same as that of Request QPS.

#### 11.2.5.3 DDL

- Schema Version: The version of the schema currently cached in each TiFlash instance.
- Schema Apply OPM: The number of TiDB `schema diff` synchronized in `apply` operations by all TiFlash instances per minute. This item includes the count of three types of `apply`: `diff apply`, `full apply`, and `failed apply`. `diff apply` is the normal process of a single `apply`. If `diff apply` fails, `failed apply` increases by 1, and TiFlash rolls back to `full apply` and pulls the latest schema information to update the schema version of TiFlash.
- Schema Internal DDL OPM: The number of specific DDL operations executed per minute in all TiFlash instances.
- Schema Apply Duration: The time used for a single `apply schema` operation in all TiFlash instances.

#### 11.2.5.4 Storage

- Write Command OPS: The number of write requests received per second by the storage layer of all TiFlash instances.
- Write Amplification: Write amplification of each TiFlash instance (the actual bytes of disk writes divided by the written bytes of logical data). `total` is the write amplification since this start, and `5min` is the write amplification in the last 5 minutes.
- Read Tasks OPS: The number of read tasks in the storage layer per second for each TiFlash instance.
- Rough Set Filter Rate: The proportion of the number of packets read by each TiFlash instance in the last minute that are filtered by the rough set index of the storage layer.
- Internal Tasks OPS: The number of times that all TiFlash instances perform internal data sorting tasks per second.
- Internal Tasks Duration: The time consumed by all TiFlash instances for internal data sorting tasks.
- Page GC Tasks OPM: The number of times that all TiFlash instances perform Delta data sorting tasks per minute.

- Page GC Tasks Duration: The distribution of time consumed by all TiFlash instances to perform Delta data sorting tasks.
- Disk Write OPS: The number of disk writes per second by all TiFlash instances.
- Disk Read OPS: The number of disk reads per second by all TiFlash instances.
- Write flow: The traffic of disk writes by all TiFlash instances.
- Read flow: The traffic of disk reads by all TiFlash instances.

**Note:**

These metrics only cover the monitoring information of the TiFlash storage layer and do not cover that in TiFlash-Proxy.

#### 11.2.5.5 Storage Write Stall

- Write & Delta Management Throughput: The throughput of write and data compaction for all instances.
  - `throughput_write` means the throughput of data synchronization through Raft.
  - `throughput_delta-management` means the throughput of data compaction.
  - `total_write` means the total bytes written since the last start.
  - `total_delta-management` means the total bytes of data compacted since the last start.
- Write Stall Duration: The stall duration of write and removing Region data (deleting ranges) by instance.
- Write Throughput By Instance: The throughput of write by instance. It includes the throughput by applying the Raft write commands and Raft snapshots.
- Write Command OPS By Instance: The total count of different kinds of commands received by instance.
  - `write_block` means the data logs synchronized through Raft.
  - `delete_range` means that some Regions are removed from or moved to this instance.
  - `ingest` means some Region snapshots are applied to this instance.

#### 11.2.5.6 Raft

- Read Index OPS: The number of times that each TiFlash instance triggers the `read_index` request per second, which equals to the number of Regions triggered.
- Read Index Duration: The time used by `read_index` for all TiFlash instances. Most time is used for interaction with the Region leader and retry.
- Wait Index Duration: The time used by `wait_index` for all TiFlash instances, namely the time used to wait until local index  $\geq$  `read_index` after the `read_index` request is received.

## 11.3 Secure

### 11.3.1 Enable TLS between TiDB Clients and Servers

Non-encrypted connection between TiDB's server and clients is allowed by default, which enables third parties that monitor channel traffic to know the data sent and received between the server and the client, including but not limited to query content, query results, and so on. If a channel is untrustworthy (such as if the client is connected to the TiDB server via a public network), then a non-encrypted connection is prone to information leakage. In this case, for security reasons, it is recommended to require an encrypted connection.

The TiDB server supports the encrypted connection based on the TLS (Transport Layer Security). The protocol is consistent with MySQL encrypted connections and is directly supported by existing MySQL clients such as MySQL Client, MySQL Shell and MySQL drivers. TLS is sometimes referred to as SSL (Secure Sockets Layer). Because the SSL protocol has [known security vulnerabilities](#), TiDB does not support SSL. TiDB supports the following protocols: TLS 1.0, TLS 1.1, TLS 1.2 and TLS 1.3.

When an encrypted connection is used, the connection has the following security properties:

- Confidentiality: the traffic plaintext is encrypted to avoid eavesdropping
- Integrity: the traffic plaintext cannot be tampered
- Authentication: (optional) the client can verify the identity of the server and the server can verify the identity of the client to avoid man-in-the-middle attacks

To use connections secured with TLS, you first need to configure the TiDB server to enable TLS. Then you need to configure the client application to use TLS. Most client libraries enable TLS automatically when the server has TLS support configured correctly.

Similar to MySQL, TiDB allows TLS and non-TLS connections on the same TCP port. For a TiDB server with TLS enabled, you can choose to securely connect to the TiDB server through an encrypted connection, or to use an unencrypted connection. You can use the following ways to require the use of secure connections:

- Configure the launch parameter `--require-secure-transport` to require secure connections to the TiDB server for all users.
- Specify `REQUIRE SSL` when you create a user (`create user`), or modify an existing user (`alter user`), which is to specify that specified users must use the encrypted connection to access TiDB. The following is an example of creating a user:

```
CREATE USER 'u1'@'%' IDENTIFIED BY 'my_random_password' REQUIRE SSL;
```

**Note:**

If the login user has configured using the [TiDB Certificate-Based Authentication for Login](#), the user is implicitly required to enable the encrypted connection to TiDB.

#### 11.3.1.1 Configure TiDB server to use secure connections

See the following descriptions about the related parameters to enable secure connections:

- **auto-tls**: enables automatic certificate generation (since v5.2.0)
- **ssl-cert**: specifies the file path of the SSL certificate
- **ssl-key**: specifies the private key that matches the certificate
- **ssl-ca**: (optional) specifies the file path of the trusted CA certificate

`auto-tls` allows secure connections but does not provide client certificate validation. For certificate validation, and to control how certificates are generated, see the advice on configuring the `ssl-cert`, `ssl-key` and `ssl-ca` variables below.

To enable secure connections with your own certificates in the TiDB server, you must specify both of the `ssl-cert` and `ssl-key` parameters in the configuration file when you start the TiDB server. You can also specify the `ssl-ca` parameter for client authentication (see [Enable authentication](#)).

All the files specified by the parameters are in PEM (Privacy Enhanced Mail) format. Currently, TiDB does not support the import of a password-protected private key, so it is required to provide a private key file without a password. If the certificate or private key is invalid, the TiDB server starts as usual, but the client cannot connect to the TiDB server through an encrypted connection.

If the certificate parameters are correct, TiDB outputs `secure connection is enabled` when started; otherwise, it outputs `secure connection is NOT ENABLED`.

For TiDB versions earlier than v5.2.0, you can use `mysql_ssl_rsa_setup --datadir ↴ =./certs` to generate certificates. The `mysql_ssl_rsa_setup` tool is a part of MySQL Server.

#### 11.3.1.2 Configure the MySQL client to use encrypted connections

The client of MySQL 5.7 or later versions attempts to establish an encrypted connection by default. If the server does not support encrypted connections, it automatically returns to unencrypted connections. The client of MySQL earlier than version 5.7 uses the unencrypted connection by default.

You can change the connection behavior of the client using the following `--ssl-mode` parameters:

- `--ssl-mode=REQUIRED`: The client requires an encrypted connection. The connection cannot be established if the server side does not support encrypted connections.
- In the absence of the `--ssl-mode` parameter: The client attempts to use an encrypted connection, but the encrypted connection cannot be established if the server side does not support encrypted connections. Then the client uses an unencrypted connection.
- `--ssl-mode=DISABLED`: The client uses an unencrypted connection.

MySQL 8.0 clients have two SSL modes in addition to this parameter:

- `--ssl-mode=VERIFY_CA`: Validates the certificate from the server against the CA that requires `--ssl-ca`.
- `--ssl-mode=VERIFY_IDENTITY`: The same as `VERIFY_CA`, but also validating whether the hostname you are connecting to matches the certificate.

For more information, see [Client-Side Configuration for Encrypted Connections](#) in MySQL.

#### 11.3.1.3 Enable authentication

If the `ssl-ca` parameter is not specified in the TiDB server or MySQL client, the client or the server does not perform authentication by default and cannot prevent man-in-the-middle attack. For example, the client might “securely” connect to a disguised client. You can configure the `ssl-ca` parameter for authentication in the server and client. Generally, you only need to authenticate the server, but you can also authenticate the client to further enhance the security.

- To authenticate the TiDB server from the MySQL client:
  1. Specify the `ssl-cert` and `ssl-key` parameters in the TiDB server.
  2. Specify the `--ssl-ca` parameter in the MySQL client.
  3. Specify the `--ssl-mode` to `VERIFY_CA` at least in the MySQL client.
  4. Make sure that the certificate (`ssl-cert`) configured in the TiDB server is signed by the CA specified by the client `--ssl-ca` parameter; otherwise, the authentication fails.
- To authenticate the MySQL client from the TiDB server:
  1. Specify the `ssl-cert`, `ssl-key`, and `ssl-ca` parameters in the TiDB server.
  2. Specify the `--ssl-cert` and `--ssl-key` parameters in the client.
  3. Make sure the server-configured certificate and the client-configured certificate are both signed by the `ssl-ca` specified by the server.
- To perform mutual authentication, meet both of the above requirements.

By default, the server-to-client authentication is optional. Even if the client does not present its certificate of identification during the TLS handshake, the TLS connection can be still established. You can also require the client to be authenticated by specifying `require → x509` when creating a user (`create user`), granting permissions (`grant`), or modifying an existing user (`alter user`). The following is an example of creating a user:

```
create user 'u1'@'%' require x509;
```

#### Note:

If the login user has configured using the [TiDB Certificate-Based Authentication for Login](#), the user is implicitly required to enable the encrypted connection to TiDB.

#### 11.3.1.4 Check whether the current connection uses encryption

Use the `SHOW STATUS LIKE "%Ssl%"`; statement to get the details of the current connection, including whether encryption is used, the encryption protocol used by encrypted connections, the TLS version number and so on.

See the following example of the result in an encrypted connection. The results change according to different TLS versions or encryption protocols supported by the client.

```
mysql> SHOW STATUS LIKE "%Ssl%";  
.....  
| Ssl_verify_mode | 5  
| Ssl_version     | TLSv1.2  
| Ssl_cipher      | ECDHE-RSA-AES128-GCM-SHA256 |  
.....
```

For the official MySQL client, you can also use the `STATUS` or `\s` statement to view the connection status:

```
mysql> \s  
...  
SSL: Cipher in use is ECDHE-RSA-AES128-GCM-SHA256  
...
```

#### 11.3.1.5 Supported TLS versions, key exchange protocols, and encryption algorithms

The TLS versions, key exchange protocols and encryption algorithms supported by TiDB are determined by the official Golang libraries.

The crypto policy for your operating system and the client library you are using might also impact the list of supported protocols and cipher suites.

#### 11.3.1.5.1 Supported TLS versions

- TLS 1.0
- TLS 1.1
- TLS 1.2
- TLS 1.3

#### 11.3.1.5.2 Supported key exchange protocols and encryption algorithms

- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256
- TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384
- TLS\_ECDHE\_RSA\_WITH\_CHACHA20\_POLY1305\_SHA256
- TLS\_ECDHE\_ECDSA\_WITH\_CHACHA20\_POLY1305\_SHA256
- TLS\_AES\_128\_GCM\_SHA256
- TLS\_AES\_256\_GCM\_SHA384
- TLS\_CHACHA20\_POLY1305\_SHA256

#### 11.3.1.6 Reload certificate, key, and CA

To replace the certificate, the key or CA, first replace the corresponding files, then execute the `ALTER INSTANCE RELOAD TLS` statement on the running TiDB instance to reload the certificate (`ssl-cert`), the key (`ssl-key`), and the CA (`ssl-ca`) from the original configuration path. In this way, you do not need to restart the TiDB instance.

The newly loaded certificate, key, and CA take effect on the connection that is established after the statement is successfully executed. The connection established before the statement execution is not affected.

#### 11.3.1.7 Monitoring

Since TiDB v5.2.0, you can use the `Ssl_server_not_after` and `Ssl_server_not_before` → status variables to monitor the start and end dates of the validity of the certificate.

```
SHOW GLOBAL STATUS LIKE 'Ssl\_\_server\_\_not\_\%';
```

Variable_name	Value
Ssl_server_not_after	Nov 28 06:42:32 2021 UTC
Ssl_server_not_before	Aug 30 06:42:32 2021 UTC

2 rows in set (0.0076 sec)

### 11.3.1.8 See also

- [Enable TLS Between TiDB Components.](#)

## 11.3.2 Enable TLS Between TiDB Components

This document describes how to enable encrypted data transmission between components within a TiDB cluster. Once enabled, encrypted transmission is used between the following components:

- TiDB and TiKV; TiDB and PD
- TiKV and PD
- TiDB Control and TiDB; TiKV Control and TiKV; PD Control and PD
- Internal communication within each TiKV, PD, TiDB cluster

Currently, it is not supported to only enable encrypted transmission of some specific components.

### 11.3.2.1 Configure and enable encrypted data transmission

1. Prepare certificates.

It is recommended to prepare a server certificate for TiDB, TiKV, and PD separately. Make sure that these components can authenticate each other. The Control tools of TiDB, TiKV, and PD can choose to share one client certificate.

You can use tools like `openssl`, `easy-rsa` and `cfssl` to generate self-signed certificates.

If you choose `openssl`, you can refer to [generating self-signed certificates](#).

2. Configure certificates.

To enable mutual authentication among TiDB components, configure the certificates of TiDB, TiKV, and PD as follows.

- TiDB

Configure in the configuration file or command-line arguments:

```
[security]
# Path of the file that contains list of trusted SSL CAs for
    ↪ connection with cluster components.
cluster-ssl-ca = "/path/to/ca.pem"
# Path of the file that contains X509 certificate in PEM format for
    ↪ connection with cluster components.
cluster-ssl-cert = "/path/to/tidb-server.pem"
# Path of the file that contains X509 key in PEM format for
    ↪ connection with cluster components.
cluster-ssl-key = "/path/to/tidb-server-key.pem"
```

- TiKV

Configure in the configuration file or command-line arguments, and set the corresponding URL to [https](https://):

```
[security]
## The path for certificates. An empty string means that secure
    ↪ connections are disabled.
# Path of the file that contains a list of trusted SSL CAs. If it
    ↪ is set, the following settings `cert_path` and `key_path`
    ↪ are also needed.
ca-path = "/path/to/ca.pem"
# Path of the file that contains X509 certificate in PEM format.
cert-path = "/path/to/tikv-server.pem"
# Path of the file that contains X509 key in PEM format.
key-path = "/path/to/tikv-server-key.pem"
```

- PD

Configure in the configuration file or command-line arguments, and set the corresponding URL to [https](https://):

```
[security]
## The path for certificates. An empty string means that secure
    ↪ connections are disabled.
# Path of the file that contains a list of trusted SSL CAs. If it
    ↪ is set, the following settings `cert_path` and `key_path`
    ↪ are also needed.
cacert-path = "/path/to/ca.pem"
# Path of the file that contains X509 certificate in PEM format.
cert-path = "/path/to/pd-server.pem"
# Path of the file that contains X509 key in PEM format.
key-path = "/path/to/pd-server-key.pem"
```

- TiFlash (New in v4.0.5)

Configure in the `tiflash.toml` file, and change the `http_port` item to `https_port`:

```
toml [security] ## The path for certificates. An empty string means
    ↵ that secure connections are disabled. # Path of the file that
    ↵ contains a list of trusted SSL CAs. If it is set, the following
    ↵ settings `cert_path` and `key_path` are also needed. ca_path
    ↵ = "/path/to/ca.pem" # Path of the file that contains X509
    ↵ certificate in PEM format. cert_path = "/path/to/tiflash-server
    ↵ .pem" # Path of the file that contains X509 key in PEM format.
    ↵ key_path = "/path/to/tiflash-server-key.pem"
```

Configure in the `tiflash-learner.toml` file:

```
[security]
# Path of the file that contains a list of trusted SSL CAs. If it
    ↵ is set, the following settings `cert_path` and `key_path`
    ↵ are also needed.
ca-path = "/path/to/ca.pem"
# Path of the file that contains X509 certificate in PEM format.
cert-path = "/path/to/tiflash-server.pem"
# Path of the file that contains X509 key in PEM format.
key-path = "/path/to/tiflash-server-key.pem"
```

- TiCDC

Configure in the command-line arguments and set the corresponding URL to `https`:

```
cdb server --pd=https://127.0.0.1:2379 --log-file=ticdc.log --addr
    ↵ =0.0.0.0:8301 --advertise-addr=127.0.0.1:8301 --ca=/path/to/
    ↵ ca.pem --cert=/path/to/ticdc-cert.pem --key=/path/to/ticdc-
    ↵ key.pem
```

Now, encrypted transmission among TiDB components is enabled.

**Note:**

After enabling encrypted transmission in a TiDB cluster, if you need to connect to the cluster using `tidb-ctl`, `tikv-ctl`, or `pd-ctl`, specify the client certificate. For example:

```
./tidb-ctl -u https://127.0.0.1:10080 --ca /path/to/ca.pem --ssl-cert /
    ↵ path/to/client.pem --ssl-key /path/to/client-key.pem
```

```
tiup ctl pd -u https://127.0.0.1:2379 --cacert /path/to/ca.pem --cert /
    ↵ path/to/client.pem --key /path/to/client-key.pem
```

```
./tikv-ctl --host="127.0.0.1:20160" --ca-path="/path/to/ca.pem" --cert-
→ path="/path/to/client.pem" --key-path="/path/to/clinet-key.pem"
```

### 11.3.2.1.1 Verify component caller's identity

The Common Name is used for caller verification. In general, the callee needs to verify the caller's identity, in addition to verifying the key, the certificates, and the CA provided by the caller. For example, TiKV can only be accessed by TiDB, and other visitors are blocked even though they have legitimate certificates.

To verify component caller's identity, you need to mark the certificate user identity using `Common Name` when generating the certificate, and to check the caller's identity by configuring the `Common Name` list for the callee.

- TiDB

Configure in the configuration file or command-line arguments:

```
[security]
cluster-verify-cn = [
    "TiDB-Server",
    "TiKV-Control",
]
```

- TiKV

Configure in the configuration file or command-line arguments:

```
[security]
cert-allowed-cn = [
    "TiDB-Server", "PD-Server", "TiKV-Control", "RawKvClient1",
]
```

- PD

Configure in the configuration file or command-line arguments:

```
[security]
cert-allowed-cn = ["TiKV-Server", "TiDB-Server", "PD-Control"]
```

- TiCDC

Configure in the command-line arguments:

```
cdc server --pd=https://127.0.0.1:2379 --log-file=ticdc.log --addr
→ =0.0.0.0:8301 --advertise-addr=127.0.0.1:8301 --ca=/path/to/ca.
→ pem --cert=/path/to/ticdc-cert.pem --key=/path/to/ticdc-key.pem
→ --cert-allowed-cn="client1,client2"
```

- TiFlash (New in v4.0.5)

Configure in the `tiflash.toml` file or command-line arguments:

```
[security]
cert_allowed_cn = ["TiKV-Server", "TiDB-Server"]
```

Configure in the `tiflash-learner.toml` file:

```
[security]
cert-allowed-cn = ["PD-Server", "TiKV-Server", "TiFlash-Server"]
```

#### 11.3.2.1.2 Reload certificates

To reload the certificates and the keys, TiDB, PD, TiKV, and all kinds of clients reread the current certificates and the key files each time a new connection is created. Currently, you cannot reload the CA certificate.

#### 11.3.2.2 See also

- [Enable TLS Between TiDB Clients and Servers](#)

#### 11.3.3 Generate Self-Signed Certificates

##### Note:

To enable TLS between clients and servers, you only need to set `auto-tls`.

This document provides an example of using `openssl` to generate a self-signed certificate. You can also generate certificates and keys that meet requirements according to your demands.

Assume that the topology of the instance cluster is as follows:

Name	Host IP	Services
node1	172.16.10.11	PD1, TiDB1
node2	172.16.10.12	PD2
node3	172.16.10.13	PD3
node4	172.16.10.14	TiKV1
node5	172.16.10.15	TiKV2
node6	172.16.10.16	TiKV3

### 11.3.3.1 Install OpenSSL

- For Debian or Ubuntu OS:

```
apt install openssl
```

- For RedHat or CentOS OS:

```
yum install openssl
```

You can also refer to OpenSSL's official [download document](#) for installation.

### 11.3.3.2 Generate the CA certificate

A certificate authority (CA) is a trusted entity that issues digital certificates. In practice, contact your administrator to issue the certificate or use a trusted CA. CA manages multiple certificate pairs. Here you only need to generate an original pair of certificates as follows.

1. Generate the root key:

```
openssl genrsa -out root.key 4096
```

2. Generate root certificates:

```
openssl req -new -x509 -days 1000 -key root.key -out root.crt
```

3. Validate root certificates:

```
openssl x509 -text -in root.crt -noout
```

### 11.3.3.3 Issue certificates for individual components

This section describes how to issue certificates for individual components.

#### 11.3.3.3.1 Certificates that might be used in the cluster

- tidb-server certificate: used by TiDB to authenticate TiDB for other components and clients
- tikv-server certificate: used by TiKV to authenticate TiKV for other components and clients
- pd-server certificate: used by PD to authenticate PD for other components and clients
- client certificate: used to authenticate the clients from PD, TiKV and TiDB, such as `pd-ctl`, `tikv-ctl`

### 11.3.3.3.2 Issue certificates to TiKV instances

To issue a certificate to a TiKV instance, perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out tikv.key 2048
```

2. Make a copy of the OpenSSL configuration template file (Refer to the actual location of your template file because it might have more than one location):

```
cp /usr/lib/ssl/openssl.cnf .
```

If you do not know the actual location, look for it in the root directory:

```
find / -name openssl.cnf
```

3. Edit `openssl.cnf`, add `req_extensions = v3_req` under the `[ req ]` field, and add `subjectAltName = @alt_names` under the `[ v3_req ]` field. Finally, create a new field and edit the information of SAN.

```
[ alt_names ]
IP.1 = 127.0.0.1
IP.2 = 172.16.10.14
IP.3 = 172.16.10.15
IP.4 = 172.16.10.16
```

4. Save the `openssl.cnf` file, and generate the certificate request file (in this step, you can also assign a Common Name to the certificate, which is used to allow the server to validate the identity of the client. Each component does not enable the validation by default, and you can enable it in the configuration file):

```
openssl req -new -key tikv.key -out tikv.csr -config openssl.cnf
```

5. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA root.crt -CAkey root.key -
    ↳ CAcreateserial -in tikv.csr -out tikv.crt -extensions v3_req -
    ↳ extfile openssl.cnf
```

6. Verify that the certificate includes the SAN field (optional):

```
openssl x509 -text -in tikv.crt -noout
```

7. Confirm that the following files exist in your current directory:

```
root.crt
tikv.crt
tikv.key
```

The process of issuing certificates for other TiDB components is similar and will not be repeated in this document.

#### 11.3.3.3 Issue certificates for clients

To issue a certificate to a client, perform the following steps:

1. Generate the private key corresponding to the certificate:

```
openssl genrsa -out client.key 2048
```

2. Generate the certificate request file (in this step, you can also assign a Common Name to the certificate, which is used to allow the server to validate the identity of the client. Each component does not enable the validation by default, and you can enable it in the configuration file):

```
openssl req -new -key client.key -out client.csr
```

3. Issue and generate the certificate:

```
openssl x509 -req -days 365 -CA root.crt -CAkey root.key -
→ CAcreateserial -in client.csr -out client.crt
```

#### 11.3.4 Encryption at Rest

##### Note:

If your cluster is deployed on AWS and uses the EBS storage, it is recommended to use the EBS encryption. See [AWS documentation - EBS Encryption](#). You are using the non-EBS storage on AWS such as the local NVMe storage, it is recommended to use encryption at rest introduced in this document.

Encryption at rest means that data is encrypted when it is stored. For databases, this feature is also referred to as TDE (transparent data encryption). This is opposed to encryption in flight (TLS) or encryption in use (rarely used). Different things could be

doing encryption at rest (SSD drive, file system, cloud vendor, etc), but by having TiKV do the encryption before storage this helps ensure that attackers must authenticate with the database to gain access to data. For example, when an attacker gains access to the physical machine, data cannot be accessed by copying files on disk.

#### 11.3.4.1 Encryption support in different TiDB components

In a TiDB cluster, different components use different encryption methods. This section introduces the encryption supports in different TiDB components such as TiKV, TiFlash, PD, and Backup & Restore (BR).

When a TiDB cluster is deployed, the majority of user data is stored on TiKV and TiFlash nodes. Some metadata is stored on PD nodes (for example, secondary index keys used as TiKV Region boundaries). To get the full benefits of encryption at rest, you need to enable encryption for all components. Backups, log files, and data transmitted over the network should also be considered when you implement encryption.

##### 11.3.4.1.1 TiKV

TiKV supports encryption at rest. This feature allows TiKV to transparently encrypt data files using [AES in CTR mode](#). To enable encryption at rest, an encryption key must be provided by the user and this key is called master key. TiKV automatically rotates data keys that it used to encrypt actual data files. Manually rotating the master key can be done occasionally. Note that encryption at rest only encrypts data at rest (namely, on disk) and not while data is transferred over network. It is advised to use TLS together with encryption at rest.

Optionally, you can use AWS KMS for both cloud and on-premises deployments. You can also supply the plaintext master key in a file.

TiKV currently does not exclude encryption keys and user data from core dumps. It is advised to disable core dumps for the TiKV process when using encryption at rest. This is not currently handled by TiKV itself.

TiKV tracks encrypted data files using the absolute path of the files. As a result, once encryption is turned on for a TiKV node, the user should not change data file paths configuration such as `storage.data-dir`, `raftstore.raftdb-path`, `rocksdb.wal-dir` and `raftdb.wal-dir`.

##### 11.3.4.1.2 TiFlash

TiFlash supports encryption at rest. Data keys are generated by TiFlash. All files (including data files, schema files, and temporary files) written into TiFlash (including TiFlash Proxy) are encrypted using the current data key. The encryption algorithms, the encryption configuration (in the `tiflash-learner.toml` file) supported by TiFlash, and the meanings of monitoring metrics are consistent with those of TiKV.

If you have deployed TiFlash with Grafana, you can check the **TiFlash-Proxy-Details -> Encryption** panel.

#### 11.3.4.1.3 PD

Encryption-at-rest for PD is an experimental feature, which is configured in the same way as in TiKV.

#### 11.3.4.1.4 Backups with BR

BR supports S3 server-side encryption (SSE) when backing up data to S3. A customer-owned AWS KMS key can also be used together with S3 server-side encryption. See [BR S3 server-side encryption](#) for details.

#### 11.3.4.1.5 Logging

TiKV, TiDB, and PD info logs might contain user data for debugging purposes. The info log and this data in it are not encrypted. It is recommended to enable [log redaction](#).

### 11.3.4.2 TiKV encryption at rest

#### 11.3.4.2.1 Overview

TiKV currently supports encrypting data using AES128, AES192 or AES256, in CTR mode. TiKV uses envelope encryption. As a result, two types of keys are used in TiKV when encryption is enabled.

- Master key. The master key is provided by user and is used to encrypt the data keys TiKV generates. Management of master key is external to TiKV.
- Data key. The data key is generated by TiKV and is the key actually used to encrypt data.

The same master key can be shared by multiple instances of TiKV. The recommended way to provide a master key in production is via AWS KMS. Create a customer master key (CMK) through AWS KMS, and then provide the CMK key ID to TiKV in the configuration file. The TiKV process needs access to the KMS CMK while it is running, which can be done by using an [IAM role](#). If TiKV fails to get access to the KMS CMK, it will fail to start or restart. Refer to AWS documentation for [KMS](#) and [IAM](#) usage.

Alternatively, if using custom key is desired, supplying the master key via file is also supported. The file must contain a 256 bits (or 32 bytes) key encoded as hex string, end with a newline (namely, `\n`), and contain nothing else. Persisting the key on disk, however, leaks the key, so the key file is only suitable to be stored on the `tmpfs` in RAM.

Data keys are passed to the underlying storage engine (namely, RocksDB). All files written by RocksDB, including SST files, WAL files, and the MANIFEST file, are encrypted by the current data key. Other temporary files used by TiKV that may include user data are also encrypted using the same data key. Data keys are automatically rotated by TiKV every week by default, but the period is configurable. On key rotation, TiKV does not rewrite

all existing files to replace the key, but RocksDB compaction are expected to rewrite old data into new data files, with the most recent data key, if the cluster gets constant write workload. TiKV keeps track of the key and encryption method used to encrypt each of the files and use the information to decrypt the content on reads.

Regardless of data encryption method, data keys are encrypted using AES256 in GCM mode for additional authentication. This required the master key to be 256 bits (32 bytes), when passing from file instead of KMS.

#### 11.3.4.2.2 Key creation

To create a key on AWS, follow these steps:

1. Go to the [AWS KMS](#) on the AWS console.
2. Make sure that you have selected the correct region on the top right corner of your console.
3. Click **Create key** and select **Symmetric** as the key type.
4. Set an alias for the key.

You can also perform the operations using the AWS CLI:

```
aws --region us-west-2 kms create-key
aws --region us-west-2 kms create-alias --alias-name "alias/tidb-tde" --
    ↪ target-key-id 0987dcba-09fe-87dc-65ba-ab0987654321
```

The `--target-key-id` to enter in the second command is in the output of the first command.

#### 11.3.4.2.3 Configure encryption

To enable encryption, you can add the encryption section in the configuration files of TiKV and PD:

```
[security.encryption]
data-encryption-method = "aes128-ctr"
data-key-rotation-period = "168h" # 7 days
```

Possible values for `data-encryption-method` are “aes128-ctr”, “aes192-ctr”, “aes256-ctr” and “plaintext”. The default value is “plaintext”, which means encryption is not turned on. `data-key-rotation-period` defines how often TiKV rotates the data key. Encryption can be turned on for a fresh TiKV cluster, or an existing TiKV cluster, though only data written after encryption is enabled is guaranteed to be encrypted. To disable encryption, remove `data-encryption-method` in the configuration file, or reset it to “plaintext”, and restart TiKV. To change encryption method, update `data-encryption-method` in the configuration file and restart TiKV.

The master key has to be specified if encryption is enabled (that is, `data-encryption-method` is not “plaintext”). To specify a AWS KMS CMK as master key, add the `encryption.master-key` section after the `encryption` section:

```
[security.encryption.master-key]
type = "kms"
key-id = "0987dcba-09fe-87dc-65ba-ab0987654321"
region = "us-west-2"
endpoint = "https://kms.us-west-2.amazonaws.com"
```

The `key-id` specifies the key ID for the KMS CMK. The `region` is the AWS region name for the KMS CMK. The `endpoint` is optional and you do not need to specify it normally unless you are using an AWS KMS-compatible service from a non-AWS vendor or need to use a [VPC endpoint for KMS](#).

You can also use [multi-Region keys](#) in AWS. For this, you need to set up a primary key in a specific region and add replica keys in the regions you require.

To specify a master key that’s stored in a file, the master key configuration would look like the following:

```
[security.encryption.master-key]
type = "file"
path = "/path/to/key/file"
```

Here `path` is the path to the key file. The file must contain a 256 bits (or 16 bytes) key encoded as hex string, end with a newline (`\n`) and contain nothing else. Example of the file content:

```
3b5896b5be691006e0f71c3040a29495ddcad20b14aff61806940ebd780d3c62
```

#### 11.3.4.2.4 Rotate the master key

To rotate master key, you have to specify both of the new master key and old master key in the configuration, and restart TiKV. Use `security.encryption.master-key` to specify the new master key, and use `security.encryption.previous-master-key` to specify the old master key. The configuration format for `security.encryption.previous-master-key` is the same as `encryption.master-key`. On restart TiKV must access both of the old and new master key, but once TiKV is up and running, TiKV will only need access to the new key. It is okay to leave the `encryption.previous-master-key` configuration in the configuration file from that on. Even on restart, TiKV only tries to use the old key if it fails to decrypt existing data using the new master key.

Currently online master key rotation is not supported, so you need to restart TiKV. It is advised to do a rolling restart to a running TiKV cluster serving online query.

Here is an example configuration for rotating the KMS CMK:

```
[security.encryption.master-key]
type = "kms"
key-id = "50a0c603-1c6f-11e6-bb9e-3fadde80ce75"
region = "us-west-2"

[security.encryption.previous-master-key]
type = "kms"
key-id = "0987dcba-09fe-87dc-65ba-ab0987654321"
region = "us-west-2"
```

#### 11.3.4.2.5 Monitoring and debugging

To monitor encryption at rest, if you deploy TiKV with Grafana, you can look at the **Encryption** panel in the **TiKV-Details** dashboard. There are a few metrics to look for:

- Encryption initialized: 1 if encryption is initialized during TiKV startup, 0 otherwise. In case of master key rotation, after encryption is initialized, TiKV do not need access to the previous master key.
- Encryption data keys: number of existing data keys. The number is bumped by 1 after each time data key rotation happened. Use this metrics to monitor if data key rotation works as expected.
- Encrypted files: number of encrypted data files currently exists. Compare this number to existing data files in the data directory to estimate portion of data being encrypted, when turning on encryption for a previously unencrypted cluster.
- Encryption meta file size: size of the encryption meta data files.
- Read/Write encryption meta duration: the extra overhead to operate on metadata for encryption.

For debugging, the `tikv-ctl` command can be used to dump encryption metadata such as encryption method and data key id used to encryption the file, as well as list of data keys. Since the operation can expose sensitive data, it is not recommended to use in production. Please refer to [TiKV Control](/tikv-control.md#dump-encryption-metadata) document.

#### 11.3.4.2.6 Compatibility between TiKV versions

To reduce the overhead caused by I/O and mutex contention when TiKV manages the encryption metadata, an optimization is introduced in TiKV v4.0.9 and controlled by `security.encryption.enable-file-dictionary-log` in the TiKV configuration file. This configuration parameter takes effect only on TiKV v4.0.9 or later versions.

When it is enabled (by default), the data format of encryption metadata is unrecognizable by TiKV v4.0.8 or earlier versions. For example, assume that you use TiKV v4.0.9 or later with encryption at rest and the default `enable-file-dictionary-log` configuration. If you downgrade your cluster to TiKV v4.0.8 or earlier, TiKV will fail to start, with an error in the info log similar to the following one:

```
[2020/12/07 07:26:31.106 +08:00] [ERROR] [mod.rs:110] ["encryption: failed
    ↪ to load file dictionary."]
[2020/12/07 07:26:33.598 +08:00] [FATAL] [lib.rs:483] ["called `Result::
    ↪ unwrap()` on an `Err` value: Other(\"[components/encryption/src/
    ↪ encrypted_file/header.rs:18]: unknown version 2\")")]
```

To avoid the error above, you can first set `security.encryption.enable-file-dictionary-log` to `false` and start TiKV with v4.0.9 or later. Once TiKV starts successfully, the data format of encryption metadata is downgraded to the version recognizable to earlier TiKV versions. At this point, you can then downgrade your TiKV cluster to an earlier version.

#### 11.3.4.3 BR S3 server-side encryption

To enable S3 server-side encryption when backup to S3 using BR, pass `--s3.sse` argument and set value to “aws:kms”. S3 will use its own KMS key for encryption. Example:

```
./br backup full --pd <pd-address> --storage "s3://<bucket>/<prefix>" --s3.
    ↪ sse aws:kms
```

To use a custom AWS KMS CMK that you created and owned, pass `--s3.sse-kms-key-id` in addition. In this case, both the BR process and all the TiKV nodes in the cluster would need access to the KMS CMK (for example, via AWS IAM), and the KMS CMK needs to be in the same AWS region as the S3 bucket used to store the backup. It is advised to grant access to the KMS CMK to BR process and TiKV nodes via AWS IAM. Refer to AWS documentation for usage of [IAM](#). For example:

```
./br backup full --pd <pd-address> --storage "s3://<bucket>/<prefix>" --s3.
    ↪ region <region> --s3.sse aws:kms --s3.sse-kms-key-id 0987dcba-09fe-87
    ↪ dc-65ba-ab0987654321
```

When restoring the backup, both `--s3.sse` and `--s3.sse-kms-key-id` should NOT be used. S3 will figure out encryption settings by itself. The BR process and TiKV nodes in the cluster to restore the backup to would also need access to the KMS CMK, or the restore will fail. Example:

```
./br restore full --pd <pd-address> --storage "s3://<bucket>/<prefix>" --s3.
    ↪ region <region>"
```

#### 11.3.5 Enable Encryption for Disk Spill

When the `oom-use-tmp-storage` configuration item is set to `true`, if the memory usage of a single SQL statement exceeds the limit of `mem-quota-query` setting, some operators can save the intermediate results during execution as a temporary file to the disk and delete the file after the query is completed.

You can enable encryption for disk spill to prevent attackers from accessing data by reading these temporary files.

#### 11.3.5.1 Configure

To enable encryption for the disk spill files, you can configure the item `spilled-file-encryption-method` in the `[security]` section of the TiDB configuration file.

```
[security]
spilled-file-encryption-method = "aes128-ctr"
```

Value options for `spilled-file-encryption-method` are `aes128-ctr` and `plaintext`. The default value is `plaintext`, which means that encryption is disabled.

#### 11.3.6 Log Redaction

When TiDB provides detailed log information, it might print sensitive data (for example, user data) in the log, which causes data security risks. To avoid such risks, each component (TiDB, TiKV, and PD) provides a configuration item that enables log redaction to shield user data values.

##### 11.3.6.1 Log redaction in TiDB side

To enable log redaction in the TiDB side, set the value of `global.tidb_redact_log` to 1. This configuration value defaults to 0, which means that log redaction is disabled.

You can use the `set` syntax to set the global variable `tidb_redact_log`:

```
set @@global.tidb_redact_log=1;
```

After the setting, all logs generated in new sessions are redacted:

```
create table t (a int, unique key (a));
Query OK, 0 rows affected (0.00 sec)

insert into t values (1),(1);
ERROR 1062 (23000): Duplicate entry '1' for key 'a'
```

The error log for the `INSERT` statement above is printed as follows:

```
[2020/10/20 11:45:49.539 +08:00] [INFO] [conn.go:800] ["command dispatched
↪ failed"] [conn=5] [connInfo="id:5, addr:127.0.0.1:57222 status:10,
↪ collation:utf8_general_ci, user:root"] [command=Query] [status="inTxn
↪ :0, autocommit:1"] [sql="insert into t values ( ? ) , ( ? )"] [
↪ txn_mode=OPTIMISTIC] [err="[kv:1062]Duplicate entry '?' for key 'a'"]
```

From the error log above, you can see that all sensitive information is shielded using ? after `tidb_redact_log` is enabled. In this way, data security risks are avoided.

### 11.3.6.2 Log redaction in TiKV side

To enable log redaction in the TiKV side, set the value of `security.redact-info-log` → to `true`. This configuration value defaults to `false`, which means that log redaction is disabled.

### 11.3.6.3 Log redaction in PD side

To enable log redaction in the PD side, set the value of `security.redact-info-log` to `true`. This configuration value defaults to `false`, which means that log redaction is disabled.

### 11.3.6.4 Log redaction in TiFlash side

To enable log redaction in the TiFlash side, set both the `security.redact_info_log` value in `tiflash-server` and the `security.redact-info-log` value in `tiflash-learner` to `true`. Both configuration values default to `false`, which means that log redaction is disabled.

## 11.4 Privileges

### 11.4.1 Security Compatibility with MySQL

TiDB supports similar security functionality to MySQL 5.7, with the following exceptions:

- Column level permissions are not supported
- Password expiry, as well as password last-changed tracking and password lifetime are not supported [#9709](#)
- The permission attributes `max_questions`, `max_updated`, `max_connections`, `max_user_connections` are not supported
- Password validation is not currently supported [#9741](#)

#### 11.4.1.1 Authentication plugin status

TiDB supports multiple authentication methods. These methods can be specified on a per user basis using `CREATE USER` and `ALTER USER`. These methods are compatible with the authentication methods of MySQL with the same names.

You can use one of the following supported authentication methods in the table. To specify a default method that the server advertises when the client-server connection is being established, set the `default_authentication_plugin` variable.

The support for TLS authentication is configured differently. For detailed information, see [Enable TLS between TiDB Clients and Servers](#).

Authentication Method	Supported
<code>mysql_native_password</code>	Yes

Authentication Method	Supported
sha256_password	No
caching_sha2_password	Yes, since 5.2.0
auth_socket	Yes, since 5.3.0
TLS Certificates	Yes
LDAP	No
PAM	No
ed25519 (MariaDB)	No
GSSAPI (MariaDB)	No
FIDO	No

## 11.4.2 Privilege Management

TiDB supports MySQL 5.7's privilege management system, including the syntax and privilege types. The following features from MySQL 8.0 are also supported:

- SQL Roles, starting with TiDB 3.0.
- Dynamic privileges, starting with TiDB 5.1.

This document introduces privilege-related TiDB operations, privileges required for TiDB operations and implementation of the privilege system.

### 11.4.2.1 Privilege-related operations

#### 11.4.2.1.1 Grant privileges

The GRANT statement grants privileges to the user accounts.

For example, use the following statement to grant the `xxx` user the privilege to read the `test` database.

```
GRANT SELECT ON test.* TO 'xxx'@'%';
```

Use the following statement to grant the `xxx` user all privileges on all databases:

```
GRANT ALL PRIVILEGES ON *.* TO 'xxx'@'%';
```

By default, GRANT statements will return an error if the user specified does not exist. This behavior depends on if the SQL Mode `NO_AUTO_CREATE_USER` is specified:

```
mysql> SET sql_mode=DEFAULT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
```

```
+---  
| @@sql_mode  
|  
+---  
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,  
|   ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION  
|  
+---  
|  
1 row in set (0.00 sec)  
  
mysql> SELECT * FROM mysql.user WHERE user='idontexist';  
Empty set (0.00 sec)  
  
mysql> GRANT ALL PRIVILEGES ON test.* TO 'idontexist';  
ERROR 1105 (HY000): You are not allowed to create a user with GRANT  
  
mysql> SELECT user,host,authentication_string FROM mysql.user WHERE user=  
|   idontexist';  
Empty set (0.00 sec)
```

In the following example, the user `idontexist` is automatically created with an empty password because the SQL Mode `NO_AUTO_CREATE_USER` was not set. This is **not recommended** since it presents a security risk: miss-spelling a username will result in a new user created with an empty password:

```
mysql> SET @@sql_mode='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,  
    → NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,  
    → NO_ENGINE_SUBSTITUTION';  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @@sql_mode;
```

6

↑

| Local mode

↑

1

```

→ -----
→
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
→ ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION |
+--+
→ -----
→
1 row in set (0.00 sec)

mysql> SELECT * FROM mysql.user WHERE user='idontexist';
Empty set (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON test.* TO 'idontexist';
Query OK, 1 row affected (0.05 sec)

mysql> SELECT user,host,authentication_string FROM mysql.user WHERE user='
→ idontexist';
+-----+-----+
| user | host | authentication_string |
+-----+-----+
| idontexist | % | |
+-----+-----+
1 row in set (0.01 sec)

```

You can use fuzzy matching in GRANT to grant privileges to databases.

```

mysql> GRANT ALL PRIVILEGES ON `te%`.* TO genius;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT user,host,db FROM mysql.db WHERE user='genius';
+-----+-----+-----+
| user | host | db |
+-----+-----+-----+
| genius | % | te% |
+-----+-----+-----+
1 row in set (0.00 sec)

```

In this example, because of the % in te%, all the databases starting with te are granted the privilege.

#### 11.4.2.1.2 Revoke privileges

The REVOKE statement enables system administrators to revoke privileges from the user accounts.

The REVOKE statement corresponds with the REVOKE statement:

```
REVOKE ALL PRIVILEGES ON `test`.* FROM 'genius'@'localhost';
```

**Note:**

To revoke privileges, you need the exact match. If the matching result cannot be found, an error will be displayed:

```
mysql> REVOKE ALL PRIVILEGES ON `te%`.* FROM 'genius'@'%';
ERROR 1141 (42000): There is no such grant defined for user 'genius' on
↪ host '%'
```

About fuzzy matching, escape, string and identifier:

```
mysql> GRANT ALL PRIVILEGES ON `te\%`.* TO 'genius'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

This example uses exact match to find the database named `te%`. Note that the `%` uses the `\` escape character so that `%` is not considered as a wildcard.

A string is enclosed in single quotation marks(`'`), while an identifier is enclosed in backticks (```). See the differences below:

```
mysql> GRANT ALL PRIVILEGES ON 'test'.* TO 'genius'@'localhost';
ERROR 1064 (42000): You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right
syntax to use near ''test'.* to 'genius'@'localhost'' at line 1

mysql> GRANT ALL PRIVILEGES ON `test`.* TO 'genius'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

If you want to use special keywords as table names, enclose them in backticks (`"`). For example:

```
mysql> CREATE TABLE `select` (id int);
Query OK, 0 rows affected (0.27 sec)
```

#### 11.4.2.1.3 Check privileges granted to users

You can use the `SHOW GRANTS` statement to see what privileges are granted to a user. For example:

```
SHOW GRANTS; -- show grants for the current user

+-----+
| Grants for User |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION |
+-----+
SHOW GRANTS FOR 'root'@'%'; -- show grants for a specific user
```

For example, create a user `rw_user@192.168.%` and grant the user with write privilege on the `test.write_table` table and global read privilege.

```
CREATE USER `rw_user`@`192.168.%`;
GRANT SELECT ON *.* TO `rw_user`@`192.168.%`;
GRANT INSERT, UPDATE ON `test`.`write_table` TO `rw_user`@`192.168.%`;
```

Show granted privileges of the `rw_user@192.168.%` user:

```
SHOW GRANTS FOR `rw_user`@`192.168.%`;

+-----+
| Grants for rw_user@192.168.% |
+-----+
| GRANT Select ON *.* TO 'rw_user'@'192.168.%' |
| GRANT Insert,Update ON test.write_table TO 'rw_user'@'192.168.%' |
+-----+
```

#### 11.4.2.1.4 Dynamic privileges

Since v5.1, TiDB features support dynamic privileges, a feature borrowed from MySQL 8.0. Dynamic privileges are intended to replace the `SUPER` privilege by implementing more fine-grained access to certain operations. For example, using dynamic privileges, system administrators can create a user account that can only perform `BACKUP` and `RESTORE` operations.

Dynamic privileges include:

- `BACKUP_ADMIN`
- `RESTORE_ADMIN`
- `ROLE_ADMIN`
- `CONNECTION_ADMIN`
- `SYSTEM_VARIABLES_ADMIN`

To see the full set of dynamic privileges, execute the `SHOW PRIVILEGES` statement. Because plugins are permitted to add new privileges, the list of privileges that are assignable might differ based on your TiDB installation.

#### 11.4.2.2 Privileges required for TiDB operations

You can check privileges of TiDB users in the INFORMATION\_SCHEMA.USER\_PRIVILEGES table. For example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.USER_PRIVILEGES WHERE grantee = ''''  
      ↪ root'@'%'';  
+-----+-----+-----+-----+  
| GRANTEE   | TABLE_CATALOG | PRIVILEGE_TYPE | IS_GRANTABLE |  
+-----+-----+-----+-----+  
| 'root'@'%' | def        | Select       | YES          |  
| 'root'@'%' | def        | Insert       | YES          |  
| 'root'@'%' | def        | Update       | YES          |  
| 'root'@'%' | def        | Delete       | YES          |  
| 'root'@'%' | def        | Create       | YES          |  
| 'root'@'%' | def        | Drop         | YES          |  
| 'root'@'%' | def        | Process      | YES          |  
| 'root'@'%' | def        | References   | YES          |  
| 'root'@'%' | def        | Alter        | YES          |  
| 'root'@'%' | def        | Show Databases | YES          |  
| 'root'@'%' | def        | Super        | YES          |  
| 'root'@'%' | def        | Execute      | YES          |  
| 'root'@'%' | def        | Index        | YES          |  
| 'root'@'%' | def        | Create User  | YES          |  
| 'root'@'%' | def        | Create Tablespace | YES          |  
| 'root'@'%' | def        | Trigger      | YES          |  
| 'root'@'%' | def        | Create View  | YES          |  
| 'root'@'%' | def        | Show View    | YES          |  
| 'root'@'%' | def        | Create Role  | YES          |  
| 'root'@'%' | def        | Drop Role    | YES          |  
| 'root'@'%' | def        | CREATE TEMPORARY TABLES | YES          |  
| 'root'@'%' | def        | LOCK TABLES  | YES          |  
| 'root'@'%' | def        | CREATE ROUTINE | YES          |  
| 'root'@'%' | def        | ALTER ROUTINE | YES          |  
| 'root'@'%' | def        | EVENT        | YES          |  
| 'root'@'%' | def        | SHUTDOWN     | YES          |  
| 'root'@'%' | def        | RELOAD        | YES          |  
| 'root'@'%' | def        | FILE          | YES          |  
| 'root'@'%' | def        | CONFIG        | YES          |  
| 'root'@'%' | def        | REPLICATION CLIENT | YES          |  
| 'root'@'%' | def        | REPLICATION SLAVE | YES          |  
+-----+-----+-----+-----+  
31 rows in set (0.00 sec)
```

#### 11.4.2.2.1 ALTER

- For all `ALTER` statements, users must have the `ALTER` privilege for the corresponding table.
- For statements except `ALTER...DROP` and `ALTER...RENAME TO`, users must have the `INSERT` and `CREATE` privileges for the corresponding table.
- For the `ALTER...DROP` statement, users must have the `DROP` privilege for the corresponding table.
- For the `ALTER...RENAME TO` statement, users must have the `DROP` privilege for the table before renaming, and the `CREATE` and `INSERT` privileges for the table after renaming.

**Note:**

In MySQL 5.7 documentation, users need `INSERT` and `CREATE` privileges to perform the `ALTER` operation on a table. But in reality for MySQL 5.7.25, only the `ALTER` privilege is required in this case. Currently, the `ALTER` privilege in TiDB is consistent with the actual behavior in MySQL.

#### 11.4.2.2.2 BACKUP

Requires the `SUPER` or `BACKUP_ADMIN` privilege.

#### 11.4.2.2.3 CREATE DATABASE

Requires the `CREATE` privilege for the database.

#### 11.4.2.2.4 CREATE INDEX

Requires the `INDEX` privilege for the table.

#### 11.4.2.2.5 CREATE TABLE

Requires the `CREATE` privilege for the table.

To execute the `CREATE TABLE...LIKE...` statement, the `SELECT` privilege for the table is required.

#### 11.4.2.2.6 CREATE VIEW

Requires the `CREATE VIEW` privilege.

**Note:**

If the current user is not the user that creates the View, both the `CREATE VIEW` and `SUPER` privileges are required.

#### 11.4.2.2.7 DROP DATABASE

Requires the DROP privilege for the table.

#### 11.4.2.2.8 DROP INDEX

Requires the INDEX privilege for the table.

#### 11.4.2.2.9 DROP TABLES

Requires the DROP privilege for the table.

#### 11.4.2.2.10 LOAD DATA

Requires the INSERT privilege for the table.

#### 11.4.2.2.11 TRUNCATE TABLE

Requires the DROP privilege for the table.

#### 11.4.2.2.12 RENAME TABLE

Requires the ALTER and DROP privileges for the table before renaming and the CREATE and INSERT privileges for the table after renaming.

#### 11.4.2.2.13 ANALYZE TABLE

Requires the INSERT and SELECT privileges for the table.

#### 11.4.2.2.14 SHOW

SHOW CREATE TABLE requires any single privilege to the table.

SHOW CREATE VIEW requires the SHOW VIEW privilege.

SHOW GRANTS requires the SELECT privilege to the mysql database. If the target user is current user, SHOW GRANTS does not require any privilege.

SHOW PROCESSLIST requires SUPER to show connections belonging to other users.

#### 11.4.2.2.15 CREATE ROLE/USER

CREATE ROLE requires the CREATE ROLE privilege.

CREATE USER requires the CREATE USER privilege.

#### 11.4.2.2.16 DROP ROLE/USER

DROP ROLE requires the DROP ROLE privilege.

DROP USER requires the CREATE USER privilege.

#### 11.4.2.2.17 ALTER USER

Requires the CREATE USER privilege.

#### 11.4.2.2.18 GRANT

Requires the GRANT privilege with the privileges granted by GRANT.

Requires additional CREATE USER privilege to create a user implicitly.

GRANT ROLE requires SUPER or ROLE\_ADMIN privilege.

#### 11.4.2.2.19 REVOKE

Requires the GRANT privilege and those privileges targeted by the REVOKE statement.

REVOKE ROLE requires SUPER or ROLE\_ADMIN privilege.

#### 11.4.2.2.20 SET GLOBAL

Requires SUPER or SYSTEM\_VARIABLES\_ADMIN privilege to set global variables.

#### 11.4.2.2.21 ADMIN

Requires SUPER privilege.

#### 11.4.2.2.22 SET DEFAULT ROLE

Requires SUPER privilege.

#### 11.4.2.2.23 KILL

Requires SUPER or CONNECTION\_ADMIN privilege to kill other user sessions.

### 11.4.2.3 Implementation of the privilege system

#### 11.4.2.3.1 Privilege table

The following system tables are special because all the privilege-related data is stored in them:

- `mysql.user` (user account, global privilege)
- `mysql.db` (database-level privilege)
- `mysql.tables_priv` (table-level privilege)
- `mysql.columns_priv` (column-level privilege; not currently supported)

These tables contain the effective range and privilege information of the data. For example, in the `mysql.user` table:

```
mysql> SELECT User,Host,Select_priv,Insert_priv FROM mysql.user LIMIT 1;
+-----+-----+-----+-----+
| User | Host | Select_priv | Insert_priv |
+-----+-----+-----+-----+
| root | %    | Y          | Y          |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

In this record, `Host` and `User` determine that the connection request sent by the `root` user from any host (%) can be accepted. `Select_priv` and `Insert_priv` mean that the user has global `Select` and `Insert` privilege. The effective range in the `mysql.user` table is global.

`Host` and `User` in `mysql.db` determine which databases users can access. The effective range is the database.

#### Note:

It is recommended to only update the privilege tables via the supplied syntax such as `GRANT`, `CREATE USER` and `DROP USER`. Making direct edits to the underlying privilege tables will not automatically update the privilege cache, leading to unpredictable behavior until `FLUSH PRIVILEGES` is executed.

#### 11.4.2.3.2 Connection verification

When the client sends a connection request, TiDB server will verify the login operation. TiDB server first checks the `mysql.user` table. If a record of `User` and `Host` matches the connection request, TiDB server then verifies the `authentication_string`.

User identity is based on two pieces of information: `Host`, the host that initiates the connection, and `User`, the user name. If the user name is not empty, the exact match of user named is a must.

`User+Host` may match several rows in `user` table. To deal with this scenario, the rows in the `user` table are sorted. The table rows will be checked one by one when the client connects; the first matched row will be used to verify. When sorting, `Host` is ranked before `User`.

#### 11.4.2.3.3 Request verification

When the connection is successful, the request verification process checks whether the operation has the privilege.

For database-related requests (INSERT, UPDATE), the request verification process first checks the user's global privileges in the `mysql.user` table. If the privilege is granted, you can access directly. If not, check the `mysql.db` table.

The `user` table has global privileges regardless of the default database. For example, the `DELETE` privilege in `user` can apply to any row, table, or database.

In the `db` table, an empty user is to match the anonymous user name. Wildcards are not allowed in the `User` column. The value for the `Host` and `Db` columns can use `%` and `_`, which can use pattern matching.

Data in the `user` and `db` tables is also sorted when loaded into memory.

The use of `%` in `tables_priv` and `columns_priv` is similar, but column value in `Db`, `Table_name` and `Column_name` cannot contain `%`. The sorting is also similar when loaded.

#### 11.4.2.3.4 Time of effect

When TiDB starts, some privilege-check tables are loaded into memory, and then the cached data is used to verify the privileges. Executing privilege management statements such as `GRANT`, `REVOKE`, `CREATE USER`, `DROP USER` will take effect immediately.

Manually editing tables such as `mysql.user` with statements such as `INSERT`, `DELETE`, `UPDATE` will not take effect immediately. This behavior is compatible with MySQL, and privilege cache can be updated with the following statement:

```
FLUSH PRIVILEGES;
```

### 11.4.3 TiDB User Account Management

This document describes how to manage a TiDB user account.

#### 11.4.3.1 User names and passwords

TiDB stores the user accounts in the table of the `mysql.user` system database. Each account is identified by a user name and the client host. Each account may have a password.

You can connect to the TiDB server using the MySQL client, and use the specified account and password to login:

```
shell> mysql --port 4000 --user xxx --password
```

Or use the abbreviation of command line parameters:

```
shell> mysql -P 4000 -u xxx -p
```

#### 11.4.3.2 Add user accounts

You can create TiDB accounts in two ways:

- By using the standard account-management SQL statements intended for creating accounts and establishing their privileges, such as `CREATE USER` and `GRANT`.
- By manipulating the privilege tables directly with statements such as `INSERT`, `UPDATE`, or `DELETE`.

It is recommended to use the account-management statements, because manipulating the privilege tables directly can lead to incomplete updates. You can also create accounts by using third party GUI tools.

```
CREATE USER [IF NOT EXISTS] user [IDENTIFIED BY 'auth_string'];
```

After you assign the password, TiDB encrypts and stores the `auth_string` in the `mysql.user` table.

```
CREATE USER 'test'@'127.0.0.1' IDENTIFIED BY 'xxx';
```

The name of a TiDB account consists of a user name and a hostname. The syntax of the account name is ‘`user_name`@‘`host_name`’.

- `user_name` is case sensitive.
- `host_name` is a hostname or IP address, which supports the wild card `%` or `_`. For example, the hostname `'%'` matches all hosts, and the hostname `'192.168.1.%'` matches all hosts in the subnet.

The host supports fuzzy matching:

```
CREATE USER 'test'@'192.168.10.%';
```

The `test` user is allowed to log in from any hosts on the `192.168.10` subnet.

If the host is not specified, the user is allowed to log in from any IP. If no password is specified, the default is empty password:

```
CREATE USER 'test';
```

Equivalent to:

```
CREATE USER 'test'@'%' IDENTIFIED BY '';
```

If the specified user does not exist, the behavior of automatically creating users depends on `sql_mode`. If the `sql_mode` includes `NO_AUTO_CREATE_USER`, the `GRANT` statement will not create users with an error returned.

For example, assume that the `sql_mode` does not include `NO_AUTO_CREATE_USER`, and you use the following `CREATE USER` and `GRANT` statements to create four accounts:

```
CREATE USER 'finley'@'localhost' IDENTIFIED BY 'some_pass';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'finley'@'localhost' WITH GRANT OPTION;
```

```
CREATE USER 'finley'@'%' IDENTIFIED BY 'some_pass';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'finley'@'%' WITH GRANT OPTION;
```

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin_pass';
```

```
GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
```

```
CREATE USER 'dummy'@'localhost';
```

To see the privileges granted for an account, use the SHOW GRANTS statement:

```
SHOW GRANTS FOR 'admin'@'localhost';
```

```
+-----+
| Grants for admin@localhost |
+-----+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----+
```

#### 11.4.3.3 Remove user accounts

To remove a user account, use the DROP USER statement:

```
DROP USER 'test'@'localhost';
```

This operation clears the user's records in the `mysql.user` table and the related records in the privilege table.

#### 11.4.3.4 Reserved user accounts

TiDB creates the '`'root'@'%'`' default account during the database initialization.

#### 11.4.3.5 Set account resource limits

Currently, TiDB does not support setting account resource limits.

#### 11.4.3.6 Assign account passwords

TiDB stores passwords in the `mysql.user` system database. Operations that assign or update passwords are permitted only to users with the `CREATE USER` privilege, or, alternatively, privileges for the `mysql` database (`INSERT` privilege to create new accounts, `UPDATE` privilege to update existing accounts).

- To assign a password when you create a new account, use `CREATE USER` and include an `IDENTIFIED BY` clause:

```
CREATE USER 'test'@'localhost' IDENTIFIED BY 'mypass';
```

- To assign or change a password for an existing account, use `SET PASSWORD FOR` or `ALTER USER`:

```
SET PASSWORD FOR 'root'@'%' = 'xxx';
```

Or:

```
ALTER USER 'test'@'localhost' IDENTIFIED BY 'mypass';
```

#### 11.4.3.7 Forget the `root` password

1. Modify the configuration file by adding `skip-grant-table` in the `security` part:

```
[security]
skip-grant-table = true
```

2. Start TiDB with the modified configuration. Use `root` to log in and then modify the password:

```
mysql -h 127.0.0.1 -P 4000 -u root
```

When the `skip-grant-table` is set, starting the TiDB process will check whether the user is an administrator of the operating system, and only the `root` user of the operating system can start the TiDB process.

#### 11.4.3.8 FLUSH PRIVILEGES

Information related to users and privileges is stored in the TiKV server, and TiDB caches this information inside the process. Generally, modification of the related information through `CREATE USER`, `GRANT`, and other statements takes effect quickly within the entire cluster. If the operation is affected by some factors such as temporarily unavailable network, the modification will take effect in about 15 minutes because TiDB will periodically reload the cache information.

If you modified the privilege tables directly, run the following command to apply changes immediately:

```
FLUSH PRIVILEGES;
```

For details, see [Privilege Management](#).

#### 11.4.4 Role-Based Access Control

The implementation of TiDB's role-based access control (RBAC) system is similar to that of MySQL 8.0. TiDB is compatible with most RBAC syntax of MySQL.

This document introduces TiDB RBAC-related operations and implementation.

##### 11.4.4.1 RBAC operations

A role is a collection of a series of privileges. You can do the following operations:

- Create a role.
- Delete a role.
- Grant a privilege to a role.
- Grant a role to another user. That user can obtain the privileges involved in the role, after enabling the role.

###### 11.4.4.1.1 Create a role

For example, you can use the following statement to create the roles `app_developer`, `app_read`, and `app_write`:

```
CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

For the role naming format and rule, see [TiDB User Account Management](#).

Roles are stored in the `mysql.user` table and the host name part of the role name (if omitted) defaults to `'%'`. The name of the role you are trying to create must be unique; otherwise, an error is reported.

To create a role, you need the `CREATE ROLE` or `CREATE USER` privilege.

###### 11.4.4.1.2 Grant a privilege to a role

The operation of granting a privilege to a role is the same with that of granting a privilege to a user. For details, see [TiDB Privilege Management](#).

For example, you can use the following statement to grant the `app_read` role the privilege to read the `app_db` database:

```
GRANT SELECT ON app_db.* TO 'app_read'@'%';
```

You can use the following statement to grant the `app_write` role the privilege to write data to the `app_db` database:

```
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write'@'%';;
```

You can use the following statement to grant the `app_developer` role all privileges on the `app_db` database:

```
GRANT ALL ON app_db.* TO 'app_developer';
```

#### 11.4.4.1.3 Grant a role to a user

Assume that a user `dev1` has the developer role with all the privileges on `app_db`; two users `read_user1` and `read_user2` have the read-only privilege on `app_db`; and a user `rw_user1` has read and write privileges on `app_db`.

Use `CREATE USER` to create the users:

```
CREATE USER 'dev1'@'localhost' IDENTIFIED BY 'dev1pass';
CREATE USER 'read_user1'@'localhost' IDENTIFIED BY 'read_user1pass';
CREATE USER 'read_user2'@'localhost' IDENTIFIED BY 'read_user2pass';
CREATE USER 'rw_user1'@'localhost' IDENTIFIED BY 'rw_user1pass';
```

Then use `GRANT` to grant roles to users

```
GRANT 'app_developer' TO 'dev1'@'localhost';
GRANT 'app_read' TO 'read_user1'@'localhost', 'read_user2'@'localhost';
GRANT 'app_read', 'app_write' TO 'rw_user1'@'localhost';
```

To grant a role to another user or revoke a role, you need the `SUPER` privilege.

Granting a role to a user does not mean enabling the role immediately. Enabling a role is another operation.

The following operations might form a “relation loop.”

```
CREATE USER 'u1', 'u2';
CREATE ROLE 'r1', 'r2';

GRANT 'u1' TO 'u1';
GRANT 'r1' TO 'r1';

GRANT 'r2' TO 'u2';
GRANT 'u2' TO 'r2';
```

TiDB supports this multi-level authorization relationship. You can use it to implement privilege inheritance.

#### 11.4.4.1.4 Check a role's privileges

You can use the `SHOW GRANTS` statement to check what privileges have been granted to the user.

To check privilege-related information of another user, you need the `SELECT` privilege on the `mysql` database.

```
SHOW GRANTS FOR 'dev1'@'localhost';
```

```
+-----+
| Grants for dev1@localhost |
+-----+
| GRANT USAGE ON *.* TO `dev1`@`localhost` |
| GRANT `app_developer`@`%` TO `dev1`@`localhost` |
+-----+
```

You can use the `USING` option in `SHOW GRANTS` to check a role's privileges:

```
SHOW GRANTS FOR 'dev1'@'localhost' USING 'app_developer';
```

```
+-----+
| Grants for dev1@localhost |
+-----+
| GRANT USAGE ON *.* TO `dev1`@`localhost` |
| GRANT ALL PRIVILEGES ON `app_db`.* TO `dev1`@`localhost` |
| GRANT `app_developer`@`%` TO `dev1`@`localhost` |
+-----+
```

```
SHOW GRANTS FOR 'rw_user1'@'localhost' USING 'app_read', 'app_write';
```

```
+-----+
→
| Grants for rw_user1@localhost |
+-----+
→
| GRANT USAGE ON *.* TO `rw_user1`@`localhost` |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `app_db`.* TO `rw_user1`@`localhost` |
| GRANT `app_read`@`%`, `app_write`@`%` TO `rw_user1`@`localhost` |
+-----+
→
```

```
SHOW GRANTS FOR 'read_user1'@'localhost' USING 'app_read';
```

```
+-----+
| Grants for read_user1@localhost          |
+-----+
| GRANT USAGE ON *.* TO `read_user1`@`localhost` |
| GRANT SELECT ON `app_db`.* TO `read_user1`@`localhost` |
| GRANT `app_read`@`%` TO `read_user1`@`localhost` |
+-----+
```

You can use SHOW GRANTS or SHOW GRANTS FOR CURRENT\_USER() to check the current user's privileges. SHOW GRANTS and SHOW GRANTS FOR CURRENT\_USER() are different in the following aspects:

- SHOW GRANTS shows the privilege of the enabled role for the current user.
- SHOW GRANTS FOR CURRENT\_USER() does not show the enabled role's privilege.

#### 11.4.4.1.5 Set the default role

After a role is granted to a user, it does not take effect immediately. Only after the user enables this role, he can use the privilege the role owns.

You can set default roles for a user. When the user logs in, the default roles are automatically enabled.

```
SET DEFAULT ROLE
{NONE | ALL | role [, role] ...}
TO user [, user]
```

For example, you can use the following statement to set default roles of `rw_user1@localhost` to `app_read` and `app_write`:

```
SET DEFAULT ROLE app_read, app_write TO 'rw_user1'@'localhost';
```

You can use the following statement to set default roles of `dev1@localhost` to all roles:

```
SET DEFAULT ROLE ALL TO 'dev1'@'localhost';
```

You can use the following statement to disable all default roles of `dev1@localhost`:

```
SET DEFAULT ROLE NONE TO 'dev1'@'localhost';
```

#### Note:

You need to grant the role to the user before you set the default role to this role.

#### 11.4.4.1.6 Enable a role in the current session

You can enable some role(s) in the current session.

```
SET ROLE {  
    DEFAULT  
    | NONE  
    | ALL  
    | ALL EXCEPT role [, role ] ...  
    | role [, role ] ...  
}
```

For example, after `rw_user1` logs in, you can use the following statement to enable roles `app_read` and `app_write` that are valid only in the current session:

```
SET ROLE 'app_read', 'app_write';
```

You can use the following statement to enable the default role of the current user:

```
SET ROLE DEFAULT
```

You can use the following statement to enable all roles granted to the current user:

```
SET ROLE ALL
```

You can use the following statement to disable all roles:

```
SET ROLE NONE
```

You can use the following statement to enable roles except `app_read`:

```
SET ROLE ALL EXCEPT 'app_read'
```

#### Note:

If you use `SET ROLE` to enable a role, this role is valid only in the current session.

#### 11.4.4.1.7 Check the current enabled role

The current user can use the `CURRENT_ROLE()` function to check which role has been enabled by the current user.

For example, you can grant default roles to `rw_user1'@'localhost`:

```
SET DEFAULT ROLE ALL TO 'rw_user1'@'localhost';
```

After `rw_user1@localhost` logs in, you can execute the following statement:

```
SELECT CURRENT_ROLE();
```

```
+-----+
| CURRENT_ROLE() |
+-----+
| `app_read`@`%`, `app_write`@`%` |
+-----+
```

```
SET ROLE 'app_read'; SELECT CURRENT_ROLE();
```

```
+-----+
| CURRENT_ROLE() |
+-----+
| `app_read`@`%` |
+-----+
```

#### 11.4.4.1.8 Revoke a role

You can use the following statement to revoke the `app_read` role granted to the users `read_user1@localhost` and `read_user2@localhost`:

```
REVOKE 'app_read' FROM 'read_user1'@'localhost', 'read_user2'@'localhost';
```

You can use the following statement to revoke the roles `app_read` and `app_write` granted to the `rw_user1@localhost` user:

```
REVOKE 'app_read', 'app_write' FROM 'rw_user1'@'localhost';
```

The operation of revoking a role from a user is atomic. If you fail to revoke a role, this operation rolls back.

#### 11.4.4.1.9 Revoke a privilege

The REVOKE statement is reverse to GRANT. You can use REVOKE to revoke the privileges of `app_write`.

```
REVOKE INSERT, UPDATE, DELETE ON app_db.* FROM 'app_write';
```

For details, see [TiDB Privilege Management](#).

#### 11.4.4.1.10 Delete a role

You can use the following statement to delete roles `app_read` and `app_write`:

```
DROP ROLE 'app_read', 'app_write';
```

This operation deletes the role records of `app_read` and `app_write` in the `mysql.user` table and related records in the authorization table, and terminates the authorization related to the two roles.

To delete a role, you need the `DROP ROLE` or `DROP USER` privilege.

#### 11.4.4.1.11 Authorization table

In addition to four system [privilege tables](#), the RBAC system introduces two new system privilege tables:

- `mysql.role_edges`: records the authorization relationship of the role and user.
- `mysql.default_roles`: records default roles of each user.

`mysql.role_edges`

`mysql.role_edges` contains the following data:

```
SELECT * FROM mysql.role_edges;
```

FROM_HOST	FROM_USER	TO_HOST	TO_USER	WITH_ADMIN_OPTION
%	r_1	%	u_1	N

1 row in set (0.00 sec)

- `FROM_HOST` and `FROM_USER` indicate the role's host name and user name respectively.
- `TO_HOST` and `TO_USER` indicate the host name and user name of the user to which a role is granted.

`mysql.default_roles`

`mysql.default_roles` shows which roles have been enabled by default for each user.

```
SELECT * FROM mysql.default_roles;
```

HOST	USER	DEFAULT_ROLE_HOST	DEFAULT_ROLE_USER
%	u_1	%	r_1
%	u_1	%	r_2

2 rows in set (0.00 sec)

- `HOST` and `USER` indicate the user's host name and user name respectively.
- `DEFAULT_ROLE_HOST` and `DEFAULT_ROLE_USER` indicate the host name and user name of the default role respectively.

#### 11.4.4.1.12 References

Because RBAC, user management, and privilege management are closely related, you can refer to operation details in the following resources:

- [TiDB Privilege Management](#)
- [TiDB User Account Management](#)

#### 11.4.5 Certificate-Based Authentication for Login

TiDB supports a certificate-based authentication method for users to log into TiDB. With this method, TiDB issues certificates to different users, uses encrypted connections to transfer data, and verifies certificates when users log in. This approach is more secure than the traditional password-based authentication method commonly used by MySQL users and is thus adopted by an increasing number of users.

To use certificate-based authentication, you might need to perform the following operations:

- Create security keys and certificates
- Configure certificates for TiDB and the client
- Configure the user certificate information to be verified when the user logs in
- Update and replace certificates

The rest of the document introduces in detail how to perform these operations.

##### 11.4.5.1 Create security keys and certificates

It is recommended that you use [OpenSSL](#) to create keys and certificates. The certificate generation process is similar to the process described in [Enable TLS Between TiDB Clients and Servers](#). The following paragraphs demonstrate on how to configure more attribute fields that need to be verified in the certificate.

###### 11.4.5.1.1 Generate CA key and certificate

1. Execute the following command to generate the CA key:

```
sudo openssl genrsa 2048 > ca-key.pem
```

The output of the above command:

```
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
```

- Execute the following command to generate the certificate corresponding to the CA key:

```
sudo openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca
→ -cert.pem
```

- Enter detailed certificate information. For example:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (e.g. city) []:San Francisco
Organization Name (e.g. company) [Internet Widgits Pty Ltd]:PingCAP Inc
→ .
Organizational Unit Name (e.g. section) []:TiDB
Common Name (e.g. server FQDN or YOUR name) []:TiDB admin
Email Address []:s@pingcap.com
```

**Note:**

In the above certificate details, texts after : are the entered information.

#### 11.4.5.1.2 Generate server key and certificate

- Execute the following command to generate the server key:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout server-
→ key.pem -out server-req.pem
```

- Enter detailed certificate information. For example:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (e.g. city) []:San Francisco
Organization Name (e.g. company) [Internet Widgits Pty Ltd]:PingCAP Inc
→ .
Organizational Unit Name (e.g. section) []:TiKV
Common Name (e.g. server FQDN or YOUR name) []:TiKV Test Server
Email Address []:k@pingcap.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

3. Execute the following command to generate the RSA key of the server:

```
sudo openssl rsa -in server-key.pem -out server-key.pem
```

The output of the above command:

```
writing RSA key
```

4. Use the CA certificate signature to generate the signed server certificate:

```
sudo openssl x509 -req -in server-req.pem -days 365000 -CA ca-cert.pem
→ -CAkey ca-key.pem -set_serial 01 -out server-cert.pem
```

The output of the above command (for example):

```
Signature ok
subject=C = US, ST = California, L = San Francisco, O = PingCAP Inc.,
→ OU = TiKV, CN = TiKV Test Server, emailAddress = k@pingcap.com
Getting CA Private Key
```

#### Note:

When you log in, TiDB checks whether the information in the `subject` section of the above output is consistent or not.

#### 11.4.5.1.3 Generate client key and certificate

After generating the server key and certificate, you need to generate the key and certificate for the client. It is often necessary to generate different keys and certificates for different users.

1. Execute the following command to generate the client key:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout client-
→ key.pem -out client-req.pem
```

2. Enter detailed certificate information. For example:

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (e.g. city) []:San Francisco
Organization Name (e.g. company) [Internet Widgits Pty Ltd]:PingCAP Inc
→ .
Organizational Unit Name (e.g. section) []:TiDB
Common Name (e.g. server FQDN or YOUR name) []:tpch-user1
Email Address []:zz@pingcap.com
```

Please enter the following 'extra' attributes  
 to be sent with your certificate request  
 A challenge password []:  
 An optional company name []:

3. Execute the following command to generate the RSA key of the client:

```
sudo openssl rsa -in client-key.pem -out client-key.pem
```

The output of the above command:

```
writing RSA key
```

4. Use the CA certificate signature to generate the client certificate:

```
sudo openssl x509 -req -in client-req.pem -days 365000 -CA ca-cert.pem  

    ↪ -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

The output of the above command (for example):

```
Signature ok  

subject=C = US, ST = California, L = San Francisco, O = PingCAP Inc.,  

    ↪ OU = TiDB, CN = tpch-user1, emailAddress = zz@pingcap.com  

Getting CA Private Key
```

#### Note:

The information of the **subject** section in the above output is used for **certificate configuration for login verification** in the **require** section.

#### 11.4.5.1.4 Verify certificate

Execute the following command to verify certificate:

```
openssl verify -CAfile ca-cert.pem server-cert.pem client-cert.pem
```

If the certificate is verified, you will see the following result:

```
server-cert.pem: OK  

client-cert.pem: OK
```

#### 11.4.5.2 Configure TiDB and the client to use certificates

After generating the certificates, you need to configure the TiDB server and the client to use the corresponding server certificate or client certificate.

#### 11.4.5.2.1 Configure TiDB to use server certificate

Modify the [security] section in the TiDB configuration file. This step specifies the directory in which the CA certificate, the server key, and the server certificate are stored. You can replace path/to/server-cert.pem, path/to/server-key.pem, path/to/ca-cert.pem with your own directory.

```
[security]
ssl-cert ="path/to/server-cert.pem"
ssl-key ="path/to/server-key.pem"
ssl-ca="path/to/ca-cert.pem"
```

Start TiDB and check logs. If the following information is displayed in the log, the configuration is successful:

```
[INFO] [server.go:264] ["secure connection is enabled"] ["client
↪ verification enabled"=true]
```

#### 11.4.5.2.2 Configure the client to use client certificate

Configure the client so that the client uses the client key and certificate for login.

Taking the MySQL client as an example, you can use the newly created client certificate, client key and CA by specifying ssl-cert, ssl-key, and ssl-ca:

```
mysql -utest -h0.0.0.0 -P4000 --ssl-cert /path/to/client-cert.new.pem --ssl-
↪ key /path/to/client-key.new.pem --ssl-ca /path/to/ca-cert.pem
```

##### Note:

/path/to/client-cert.new.pem, /path/to/client-key.new.pem, and /  
↪ path/to/ca-cert.pem are the directory of the CA certificate, client key,  
and client certificate. You can replace them with your own directory.

#### 11.4.5.3 Configure the user certificate information for login verification

First, connect TiDB using the client to configure the login verification. Then, you can get and configure the user certificate information to be verified.

##### 11.4.5.3.1 Get user certificate information

The user certificate information can be specified by require subject, require issuer, require san, and require cipher, which are used to check the X509 certificate attributes.

- **require subject:** Specifies the **subject** information of the client certificate when you log in. With this option specified, you do not need to configure **require ssl** or **x509**. The information to be specified is consistent with the entered **subject** information in [Generate client keys and certificates](#).

To get this option, execute the following command:

```
openssl x509 -noout -subject -in client-cert.pem | sed 's/.\\{8\\}//' |
→ sed 's/, /\\//g' | sed 's/ = /=g' | sed 's/^\\//'
```

- **require issuer:** Specifies the **subject** information of the CA certificate that issues the user certificate. The information to be specified is consistent with the entered **subject** information in [Generate CA key and certificate](#).

To get this option, execute the following command:

```
openssl x509 -noout -subject -in ca-cert.pem | sed 's/.\\{8\\}//' | sed ' |
→ s/, /\\//g' | sed 's/ = /=g' | sed 's/^\\//'
```

- **require san:** Specifies the Subject Alternative Name information of the CA certificate that issues the user certificate. The information to be specified is consistent with the [alt\\_names of the openssl.cnf configuration file](#) used to generate the client certificate.

- Execute the following command to get the information of the **require san** item in the generated certificate:

```
openssl x509 -noout -ext subjectAltName -in client.crt
```

- **require san** currently supports the following Subject Alternative Name check items:

- \* URI
- \* IP
- \* DNS

- Multiple check items can be configured after they are connected by commas. For example, configure **require san** as follows for the **u1** user:

```
create user 'u1'@'%' require san 'DNS:d1,URI:spiffe://example.org/
→ myservice1,URI:spiffe://example.org/myservice2'
```

The above configuration only allows the **u1** user to log in to TiDB using the certificate with the **URI** item **spiffe://example.org/myservice1** or **spiffe://example.org/myservice2** and the **DNS** item **d1**.

- **require cipher:** Checks the cipher method supported by the client. Use the following statement to check the list of supported cipher methods:

```
SHOW SESSION STATUS LIKE 'Ssl_cipher_list';
```

#### 11.4.5.3.2 Configure user certificate information

After getting the user certificate information (`require subject`, `require issuer`, `require san`, `require cipher`), configure these information to be verified when creating a user, granting privileges, or altering a user. Replace `<replaceable>` with the corresponding information in the following statements.

You can configure one option or multiple options using the space or `and` as the separator.

- Configure user certificate when creating a user (`create user`):

```
create user 'u1'@'%' require issuer '<replaceable>' subject '<
↪ replaceable>' san '<replaceable>' cipher '<replaceable>';
```

- Configure user certificate when granting privileges:

```
grant all on *.* to 'u1'@'%' require issuer '<replaceable>' subject '<
↪ replaceable>' san '<replaceable>' cipher '<replaceable>';
```

- Configure user certificate when altering a user:

```
alter user 'u1'@'%' require issuer '<replaceable>' subject '<
↪ replaceable>' san '<replaceable>' cipher '<replaceable>';
```

After the above configuration, the following items will be verified when you log in:

- SSL is used; the CA that issues the client certificate is consistent with the CA configured in the server.
- The `issuer` information of the client certificate matches the information specified in `require issuer`.
- The `subject` information of the client certificate matches the information specified in `require cipher`.
- The `Subject Alternative Name` information of the client certificate matches the information specified in `require san`.

You can log into TiDB only after all the above items are verified. Otherwise, the `ERROR 1045 (28000): Access denied` error is returned. You can use the following command to check the TLS version, the cipher algorithm and whether the current connection uses the certificate for the login.

Connect the MySQL client and execute the following statement:

```
\s
```

The output:

```
-----
mysql Ver 15.1 Distrib 10.4.10-MariaDB, for Linux (x86_64) using readline
→ 5.1

Connection id:      1
Current database:  test
Current user:      root@127.0.0.1
SSL:               Cipher in use is TLS_AES_256_GCM_SHA384
```

Then execute the following statement:

```
show variables like '%ssl%';
```

The output:

Variable_name	Value
ssl_cert	/path/to/server-cert.pem
ssl_ca	/path/to/ca-cert.pem
have_ssl	YES
have_openssl	YES
ssl_key	/path/to/server-key.pem

6 rows in set (0.067 sec)

#### 11.4.5.4 Update and replace certificate

The key and certificate are updated regularly. The following sections introduce how to update the key and certificate.

The CA certificate is the basis for mutual verification between the client and server. To replace the CA certificate, generate a combined certificate that supports the authentication for both old and new certificates. On the client and server, first replace the CA certificate, then replace the client/server key and certificate.

##### 11.4.5.4.1 Update CA key and certificate

- Back up the old CA key and certificate (suppose that `ca-key.pem` is stolen):

```
mv ca-key.pem ca-key.old.pem && \
mv ca-cert.pem ca-cert.old.pem
```

- Generate the new CA key:

```
sudo openssl genrsa 2048 > ca-key.pem
```

3. Generate the new CA certificate using the newly generated CA key:

```
sudo openssl req -new -x509 -nodes -days 365000 -key ca-key.pem -out ca
→ -cert.new.pem
```

**Note:**

Generating the new CA certificate is to replace the keys and certificates on the client and server, and to ensure that online users are not affected. Therefore, the appended information in the above command must be consistent with the `require issuer` information.

4. Generate the combined CA certificate:

```
cat ca-cert.new.pem ca-cert.old.pem > ca-cert.pem
```

After the above operations, restart the TiDB server with the newly created combined CA certificate. Then the server accepts both the new and old CA certificates.

Also replace the old CA certificate with the combined certificate so that the client accepts both the old and new CA certificates.

#### 11.4.5.4.2 Update client key and certificate

**Note:**

Perform the following steps **only after** you have replaced the old CA certificate on the client and server with the combined CA certificate.

1. Generate the new RSA key of the client:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout client-
→ key.new.pem -out client-req.new.pem && \
sudo openssl rsa -in client-key.new.pem -out client-key.new.pem
```

**Note:**

The above command is to replace the client key and certificate, and to ensure that the online users are not affected. Therefore, the appended information in the above command must be consistent with the `require ↵ subject` information.

2. Use the combined certificate and the new CA key to generate the new client certificate:

```
sudo openssl x509 -req -in client-req.new.pem -days 365000 -CA ca-cert.
    ↵ pem -CAkey ca-key.pem -set_serial 01 -out client-cert.new.pem
```

3. Make the client (for example, MySQL) connect TiDB with the new client key and certificate:

```
mysql -utest -h0.0.0.0 -P4000 --ssl-cert /path/to/client-cert.new.pem
    ↵ --ssl-key /path/to/client-key.new.pem --ssl-ca /path/to/ca-cert.
    ↵ pem
```

**Note:**

`/path/to/client-cert.new.pem`, `/path/to/client-key.new.pem`, and `/path/to/ca-cert.pem` specify the directory of the CA certificate, client key, and client certificate. You can replace them with your own directory.

#### 11.4.5.4.3 Update the server key and certificate

1. Generate the new RSA key of the server:

```
sudo openssl req -newkey rsa:2048 -days 365000 -nodes -keyout server-
    ↵ key.new.pem -out server-req.new.pem && \
sudo openssl rsa -in server-key.new.pem -out server-key.new.pem
```

2. Use the combined CA certificate and the new CA key to generate the new server certificate:

```
sudo openssl x509 -req -in server-req.new.pem -days 365000 -CA ca-cert.
    ↵ pem -CAkey ca-key.pem -set_serial 01 -out server-cert.new.pem
```

3. Configure the TiDB server to use the new server key and certificate. See [Configure TiDB server](#) for details.

## 11.5 SQL

### 11.5.1 SQL Language Structure and Syntax

#### 11.5.1.1 Attributes

##### 11.5.1.1.1 AUTO\_INCREMENT

This document introduces the `AUTO_INCREMENT` column attribute, including its concept, implementation principles, auto-increment related features, and restrictions.

###### Concept

`AUTO_INCREMENT` is a column attribute that is used to automatically fill in default column values. When the `INSERT` statement does not specify values for the `AUTO_INCREMENT` column, the system automatically assigns values to this column.

For performance reasons, `AUTO_INCREMENT` numbers are allocated in a batch of values (30 thousand by default) to each TiDB server. This means that while `AUTO_INCREMENT` numbers are guaranteed to be unique, values assigned to an `INSERT` statement will only be monotonic on a per TiDB server basis.

The following is a basic example of `AUTO_INCREMENT`:

```
CREATE TABLE t(id int PRIMARY KEY AUTO_INCREMENT, c int);
```

```
INSERT INTO t(c) VALUES (1);
INSERT INTO t(c) VALUES (2);
INSERT INTO t(c) VALUES (3), (4), (5);
```

```
mysql> SELECT * FROM t;
+----+---+
| id | c |
+----+---+
| 1  | 1 |
| 2  | 2 |
| 3  | 3 |
| 4  | 4 |
| 5  | 5 |
+----+---+
5 rows in set (0.01 sec)
```

In addition, `AUTO_INCREMENT` also supports the `INSERT` statements that explicitly specify column values. In such cases, TiDB stores the explicitly specified values:

```
INSERT INTO t(id, c) VALUES (6, 6);
```

```
mysql> SELECT * FROM t;
+---+---+
| id | c |
+---+---+
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
+---+---+
6 rows in set (0.01 sec)
```

The usage above is the same as that of `AUTO_INCREMENT` in MySQL. However, in terms of the specific value that is implicitly assigned, TiDB differs from MySQL significantly.

#### Implementation principles

TiDB implements the `AUTO_INCREMENT` implicit assignment in the following way:

For each auto-increment column, a globally visible key-value pair is used to record the maximum ID that has been assigned. In a distributed environment, communication between nodes has some overhead. Therefore, to avoid the issue of write amplification, each TiDB node applies for a batch of consecutive IDs as caches when assigning IDs, and then applies for the next batch of IDs after the first batch is assigned. Therefore, TiDB nodes do not apply to the storage node for IDs when assigning IDs each time. For example:

```
CREATE TABLE t(id int UNIQUE KEY AUTO_INCREMENT, c int);
```

Assume two TiDB instances, A and B, in the cluster. If you execute an `INSERT` statement on the `t` table on A and B respectively:

```
INSERT INTO t (c) VALUES (1)
```

Instance A might cache the auto-increment IDs of [1,30000], and instance B might cache the auto-increment IDs of [30001,60000]. In `INSERT` statements to be executed, these cached IDs of each instance will be assigned to the `AUTO_INCREMENT` column as the default values.

#### Basic Features

#### Uniqueness

#### Warning:

When the cluster has multiple TiDB instances, if the table schema contains the auto-increment IDs, it is recommended not to use explicit insert and

implicit assignment at the same time, which means using the default values of the auto-increment column and the custom values. Otherwise, it might break the uniqueness of implicitly assigned values.

In the example above, perform the following operations in order:

1. The client inserts a statement `INSERT INTO t VALUES (2, 1)` to instance B, which sets `id` to 2. The statement is successfully executed.
2. The client sends a statement `INSERT INTO t (c)(1)` to instance A. This statement does not specify the value of `id`, so the ID is assigned by A. At present, because A caches the IDs of [1, 30000], it might assign 2 as the value of the auto-increment ID, and increases the local counter by 1. At this time, the data whose ID is 2 already exists in the database, so the `Duplicated Error` error is returned.

### Monotonicity

TiDB guarantees that `AUTO_INCREMENT` values are monotonic (always increasing) on a per-server basis. Consider the following example where consecutive `AUTO_INCREMENT` values of 1-3 are generated:

```
CREATE TABLE t (a int PRIMARY KEY AUTO_INCREMENT, b timestamp NOT NULL
    ↪ DEFAULT NOW());
INSERT INTO t (a) VALUES (NULL), (NULL), (NULL);
SELECT * FROM t;
```

Query OK, 0 rows affected (0.11 sec)

Query OK, 3 rows affected (0.02 sec)

Records: 3 Duplicates: 0 Warnings: 0

a	b
1	2020-09-09 20:38:22
2	2020-09-09 20:38:22
3	2020-09-09 20:38:22

3 rows in set (0.00 sec)

Monotonicity is not the same guarantee as consecutive. Consider the following example:

```

CREATE TABLE t (id INT NOT NULL PRIMARY KEY auto_increment, a VARCHAR(10),
    ↪ cnt INT NOT NULL DEFAULT 1, UNIQUE KEY (a));
INSERT INTO t (a) VALUES ('A'), ('B');
SELECT * FROM t;
INSERT INTO t (a) VALUES ('A'), ('C') ON DUPLICATE KEY UPDATE cnt = cnt +
    ↪ 1;
SELECT * FROM t;

```

```
Query OK, 0 rows affected (0.00 sec)
```

```
Query OK, 2 rows affected (0.00 sec)
```

```
Records: 2 Duplicates: 0 Warnings: 0
```

id	a	cnt
1	A	1
2	B	1

```
2 rows in set (0.00 sec)
```

```
Query OK, 3 rows affected (0.00 sec)
```

```
Records: 2 Duplicates: 1 Warnings: 0
```

id	a	cnt
1	A	2
2	B	1
4	C	1

```
3 rows in set (0.00 sec)
```

In this example, the `AUTO_INCREMENT` value of 3 is allocated for the `INSERT` of the key A in `INSERT INTO t (a)VALUES ('A'), ('C')ON DUPLICATE KEY UPDATE cnt = cnt + ↪ 1;` but never used because this `INSERT` statement contains a duplicate key A. This leads to a gap where the sequence is non-consecutive. This behavior is considered legal, even though it differs from MySQL. MySQL will also have gaps in the sequence in other scenarios such as transactions being aborted and rolled back.

#### AUTO\_ID\_CACHE

The `AUTO_INCREMENT` sequence might appear to *jump* dramatically if an `INSERT` operation is performed against a different TiDB server. This is caused by the fact that each server has its own cache of `AUTO_INCREMENT` values:

```
CREATE TABLE t (a int PRIMARY KEY AUTO_INCREMENT, b timestamp NOT NULL
    ↪ DEFAULT NOW());
INSERT INTO t (a) VALUES (NULL), (NULL), (NULL);
INSERT INTO t (a) VALUES (NULL);
SELECT * FROM t;
```

Query OK, 1 row affected (0.03 sec)

a	b
1	2020-09-09 20:38:22
2	2020-09-09 20:38:22
3	2020-09-09 20:38:22
2000001	2020-09-09 20:43:43

4 rows in set (0.00 sec)

A new INSERT operation against the initial TiDB server generates the AUTO\_INCREMENT value of 4. This is because the initial TiDB server still has space left in the AUTO\_INCREMENT cache for allocation. In this case, the sequence of values cannot be considered globally monotonic, because the value of 4 is inserted after the value of 2000001:

```
mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t ORDER BY b;
+-----+
| a      | b          |
+-----+
| 1      | 2020-09-09 20:38:22 |
| 2      | 2020-09-09 20:38:22 |
| 3      | 2020-09-09 20:38:22 |
| 2000001 | 2020-09-09 20:43:43 |
| 4      | 2020-09-09 20:44:43 |
+-----+
5 rows in set (0.00 sec)
```

The AUTO\_INCREMENT cache does not persist across TiDB server restarts. The following INSERT statement is performed after the initial TiDB server is restarted:

```
mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t ORDER BY b;
```

```
+-----+
| a      | b
+-----+
| 1 | 2020-09-09 20:38:22 |
| 2 | 2020-09-09 20:38:22 |
| 3 | 2020-09-09 20:38:22 |
| 2000001 | 2020-09-09 20:43:43 |
| 4 | 2020-09-09 20:44:43 |
| 2030001 | 2020-09-09 20:54:11 |
+-----+
6 rows in set (0.00 sec)
```

A high rate of TiDB server restarts might contribute to the exhaustion of AUTO\_INCREMENT values. In the above example, the initial TiDB server still has values [5-30000] free in its cache. These values are lost, and will not be reallocated.

It is not recommended to rely on AUTO\_INCREMENT values being continuous. Consider the following example, where a TiDB server has a cache of values [2000001-2030000]. By manually inserting the value 2029998, you can see the behavior as a new cache range is retrieved:

```
mysql> INSERT INTO t (a) VALUES (2029998);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO t (a) VALUES (NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t ORDER BY b;
+-----+
| a      | b
+-----+
| 1 | 2020-09-09 20:38:22 |
| 2 | 2020-09-09 20:38:22 |
| 3 | 2020-09-09 20:38:22 |
| 2000001 | 2020-09-09 20:43:43 |
| 4 | 2020-09-09 20:44:43 |
| 2030001 | 2020-09-09 20:54:11 |
+-----+
```

```

| 2029998 | 2020-09-09 21:08:11 |
| 2029999 | 2020-09-09 21:08:11 |
| 2030000 | 2020-09-09 21:08:11 |
| 2060001 | 2020-09-09 21:08:11 |
| 2060002 | 2020-09-09 21:08:11 |
+-----+
11 rows in set (0.00 sec)

```

After the value 2030000 is inserted, the next value is 2060001. This jump in sequence is due to another TiDB server obtaining the intermediate cache range of [2030001–2060000] → . When multiple TiDB servers are deployed, there will be gaps in the AUTO\_INCREMENT sequence because cache requests are interleaved.

#### Cache size control

In earlier versions of TiDB, the cache size of the auto-increment ID was transparent to users. Starting from v3.0.14, v3.1.2, and v4.0.rc-2, TiDB has introduced the AUTO\_ID\_CACHE table option to allow users to set the cache size for allocating the auto-increment ID.

```

mysql> CREATE TABLE t(a int AUTO_INCREMENT key) AUTO_ID_CACHE 100;
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO t values();
Query OK, 1 row affected (0.00 sec)
Records: 1 Duplicates: 0 Warnings: 0

mysql> SELECT * FROM t;
+---+
| a |
+---+
| 1 |
+---+
1 row in set (0.01 sec)

```

At this time, if you invalidate the auto-increment cache of this column and redo the implicit insertion, the result is as follows:

```

mysql> DELETE FROM t;
Query OK, 1 row affected (0.01 sec)

mysql> RENAME TABLE t to t1;
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 values()
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t;

```

```
+-----+
| a   |
+-----+
| 101 |
+-----+
1 row in set (0.00 sec)
```

The re-assigned value is 101. This shows that the size of cache for allocating the auto-increment ID is 100.

In addition, when the length of consecutive IDs in a batch `INSERT` statement exceeds the length of `AUTO_ID_CACHE`, TiDB increases the cache size accordingly to ensure that the statement can be inserted properly.

#### Auto-increment step size and offset

Starting from v3.0.9 and v4.0.0-rc.1, similar to the behavior of MySQL, the value implicitly assigned to the auto-increment column is controlled by the `@@auto_increment_increment` and `@@auto_increment_offset` session variables.

The value (ID) implicitly assigned to auto-increment columns satisfies the following equation:

$$(ID - \text{auto\_increment\_offset}) \% \text{auto\_increment\_increment} == 0$$

#### Restrictions

Currently, `AUTO_INCREMENT` has the following restrictions when used in TiDB:

- It must be defined on the column of the primary key or unique index.
- It must be defined on the column of `INTEGER`, `FLOAT`, or `DOUBLE` type.
- It cannot be specified on the same column with the `DEFAULT` column value.
- `ALTER TABLE` cannot be used to add the `AUTO_INCREMENT` attribute.
- `ALTER TABLE` can be used to remove the `AUTO_INCREMENT` attribute. However, starting from v2.1.18 and v3.0.4, TiDB uses the session variable `@@tidb_allow_remove_auto_inc` to control whether `ALTER TABLE MODIFY` or `ALTER TABLE CHANGE` can be used to remove the `AUTO_INCREMENT` attribute of a column. By default, you cannot use `ALTER TABLE MODIFY` or `ALTER TABLE CHANGE` to remove the `AUTO_INCREMENT` attribute.

#### 11.5.1.1.2 AUTO\_RANDOM New in v3.1.0

##### Note:

`AUTO_RANDOM` was marked as stable in v4.0.3.

## User scenario

When you write data intensively into TiDB and TiDB has the table with a primary key of the auto-increment integer type, hotspot issue might occur. To solve the hotspot issue, you can use the `AUTO_RANDOM` attribute. Refer to [Highly Concurrent Write Best Practices](#) for details.

Take the following created table as an example:

```
CREATE TABLE t (a bigint PRIMARY KEY AUTO_INCREMENT, b varchar(255))
```

On this `t` table, you execute a large number of `INSERT` statements that do not specify the values of the primary key as below:

```
INSERT INTO t(b) VALUES ('a'), ('b'), ('c')
```

In the above statement, values of the primary key (column `a`) are not specified, so TiDB uses the continuous auto-increment row values as the row IDs, which might cause write hotspot in a single TiKV node and affect the performance. To avoid such write hotspot, you can specify the `AUTO_RANDOM` attribute rather than the `AUTO_INCREMENT` attribute for the column `a` when you create the table. See the follow examples:

```
CREATE TABLE t (a bigint PRIMARY KEY AUTO_RANDOM, b varchar(255))
```

or

```
CREATE TABLE t (a bigint AUTO_RANDOM, b varchar(255), PRIMARY KEY (a))
```

Then execute the `INSERT` statement such as `INSERT INTO t(b)VALUES....`. Now the results will be as follows:

- Implicitly allocating values: If the `INSERT` statement does not specify the values of the integer primary key column (column `a`) or specify the value as `NULL`, TiDB automatically allocates values to this column. These values are not necessarily auto-increment or continuous but are unique, which avoids the hotspot problem caused by continuous row IDs.
- Explicitly inserting values: If the `INSERT` statement explicitly specifies the values of the integer primary key column, TiDB saves these values, which works similarly to the `AUTO_INCREMENT` attribute. Note that if you do not set `NO_AUTO_VALUE_ON_ZERO` in the `@@sql_mode` system variable, TiDB will automatically allocate values to this column even if you explicitly specify the value of the integer primary key column as `0`.

### Note:

Since v4.0.3, if you want to insert values explicitly, set the value of the `@@allow_auto_random_explicit_insert` system variable to 1 (0 by default). This explicit insertion is not supported by default and the reason is documented in the [restrictions](#) section.

TiDB automatically allocates values in the following way:

The highest five digits (ignoring the sign bit) of the row value in binary (namely, shard bits) are determined by the starting time of the current transaction. The remaining digits are allocated values in an auto-increment order.

To use different number of shard bits, append a pair of parentheses to `AUTO_RANDOM` and specify the desired number of shard bits in the parentheses. See the following example:

```
CREATE TABLE t (a bigint PRIMARY KEY AUTO_RANDOM(3), b varchar(255))
```

In the above `CREATE TABLE` statement, 3 shard bits are specified. The range of the number of shard bits is `[1, field_max_bits]`. `field_max_bits` is the length of bits occupied by the primary key column.

After creating the table, use the `SHOW WARNINGS` statement to see the maximum number of implicit allocations supported by the current table:

```
SHOW WARNINGS
```

Level	Code	Message
Note	1105	Available implicit allocation times: 1152921504606846976

#### Note:

Since v4.0.3, the type of the `AUTO_RANDOM` column can only be `BIGINT`. This is to ensure the maximum number of implicit allocations.

In addition, to view the shard bit number of the table with the `AUTO_RANDOM` attribute, you can see the value of the `PK_AUTO_RANDOM_BITS=x` mode in the `TIDB_ROW_ID_SHARDING_INFO` column in the `information_schema.tables` system table. `x` is the number of shard bits.

Values allocated to the `AUTO_RANDOM` column affect `last_insert_id()`. You can use `SELECT last_insert_id ()` to get the ID that TiDB last implicitly allocates. For example:

```
INSERT INTO t (b) VALUES ("b")
SELECT * FROM t;
SELECT last_insert_id()
```

You might see the following result:

a	b
1073741825	
last_insert_id()	
1073741825	

## Compatibility

TiDB supports parsing the version comment syntax. See the following example:

```
CREATE TABLE t (a bigint PRIMARY KEY /*T! [auto_rand] auto_random */)
```

```
CREATE TABLE t (a bigint PRIMARY KEY AUTO_RANDOM)
```

The above two statements have the same meaning.

In the result of SHOW CREATE TABLE, the AUTO\_RANDOM attribute is commented out. This comment includes an attribute identifier (for example, /\*T! [auto\_rand] auto\_random \*/  
→). Here `auto_rand` represents the `AUTO_RANDOM` attribute. Only the version of TiDB that implements the feature corresponding to this identifier can parse the SQL statement fragment properly.

This attribute supports forward compatibility, namely, downgrade compatibility. TiDB of earlier versions that do not implement this feature ignore the `AUTO_RANDOM` attribute of a table (with the above comment) and can also use the table with the attribute.

## Restrictions

Pay attention to the following restrictions when you use `AUTO_RANDOM`:

- Specify this attribute for the primary key column **ONLY** of integer type. Otherwise, an error might occur. In addition, when the attribute of the primary key is `NONCLUSTERED`, `AUTO_RANDOM` is not supported even on the integer primary key. For more details about the primary key of the `CLUSTERED` type, refer to [clustered index](#).
- You cannot use `ALTER TABLE` to modify the `AUTO_RANDOM` attribute, including adding or removing this attribute.

- You cannot change the column type of the primary key column that is specified with `AUTO_RANDOM` attribute.
- You cannot specify `AUTO_RANDOM` and `AUTO_INCREMENT` for the same column at the same time.
- You cannot specify `AUTO_RANDOM` and `DEFAULT` (the default value of a column) for the same column at the same time.
- It is **not** recommended that you explicitly specify a value for the column with the `AUTO_RANDOM` attribute when you insert data. Otherwise, the numeral values that can be automatically allocated for this table might be used up in advance.

#### 11.5.1.1.3 SHARD\_ROW\_ID\_BITS

This document introduces the `SHARD_ROW_ID_BITS` table attribute, which is used to set the number of bits of the shards after the implicit `_tidb_rowid` is sharded.

##### Concept

For the tables with a non-integer primary key or no primary key, TiDB uses an implicit auto-increment row ID. When a large number of `INSERT` operations are performed, the data is written into a single Region, causing a write hot spot.

To mitigate the hot spot issue, you can configure `SHARD_ROW_ID_BITS`. The row IDs are scattered and the data are written into multiple different Regions. But setting an overlarge value might lead to an excessively large number of RPC requests, which increases the CPU and network overheads.

- `SHARD_ROW_ID_BITS = 4` indicates 16 shards
- `SHARD_ROW_ID_BITS = 6` indicates 64 shards
- `SHARD_ROW_ID_BITS = 0` indicates the default 1 shard

##### Examples

- CREATE TABLE: `CREATE TABLE t (c int)SHARD_ROW_ID_BITS = 4;`
- ALTER TABLE: `ALTER TABLE t SHARD_ROW_ID_BITS = 4;`

#### 11.5.1.2 Literal Values

TiDB literal values include character literals, numeric literals, time and date literals, hexadecimal, binary literals, and NULL literals. This document introduces each of these literal values.

This document describes String literals, Numeric literals, NULL values, Hexadecimal literals, Date and time literals, Boolean literals, and Bit-value literals.

#### 11.5.1.2.1 String literals

A string is a sequence of bytes or characters, enclosed within either single quote ' or double quote " characters. For example:

```
'example string'  
"example string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'  
'a' ' ' 'string'  
"a" ' ' "string"
```

If the `ANSI_QUOTES` SQL MODE is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

The string is divided into the following two types:

- Binary string: It consists of a sequence of bytes, whose charset and collation are both `binary`, and uses `byte` as the unit when compared with each other.
- Non-binary string: It consists of a sequence of characters and has various charsets and collations other than `binary`. When compared with each other, non-binary strings use `characters` as the unit. A character might contain multiple bytes, depending on the charset.

A string literal may have an optional `character set introducer` and `COLLATE clause`, to designate it as a string that uses a specific character set and collation.

```
[_charset_name]'string' [COLLATE collation_name]
```

For example:

```
SELECT _latin1'string';  
SELECT _binary'string';  
SELECT _utf8'string' COLLATE utf8_bin;
```

You can use N'literal' (or n'literal') to create a string in the national character set. The following statements are equivalent:

```
SELECT N'some text';  
SELECT n'some text';  
SELECT _utf8'some text';
```

To represent some special characters in a string, you can use escape characters to escape:

Escape Characters	Meaning
\0	An ASCII NUL (X'00') character
\'	A single quote ' character
\“	A double quote " character
\b	A backspace character
\n	A line break (newline) character
\r	A carriage return character
\t	A tab character
\z	ASCII 26 (Ctrl + Z)
\\\	A backslash \ character
\%	A % character
\_	A _ character

If you want to represent " in the string surrounded by ', or ' in the string surrounded by ", you do not need to use escape characters.

For more information, see [String Literals in MySQL](#).

### 11.5.1.2.2 Numeric literals

Numeric literals include integer and DECIMAL literals and floating-point literals.

Integer may include . as a decimal separator. Numbers may be preceded by - or + to indicate a negative or positive value respectively.

Exact-value numeric literals can be represented as 1, .2, 3.4, -5, -6.78, +9.10.

Numeric literals can also be represented in scientific notation, such as 1.2E3, 1.2E-3, ↵ -1.2E3, -1.2E-3.

For more information, see [Numeric Literals in MySQL](#).

### 11.5.1.2.3 Date and time literals

Date and time literal values can be represented in several formats, such as quoted strings or as numbers. When TiDB expects a date, it interprets any of '2017-08-24', '20170824' and 20170824 as a date.

TiDB supports the following date formats:

- 'YYYY-MM-DD' or 'YY-MM-DD': The - delimiter here is not strict. It can be any punctuation. For example, '2017-08-24', '2017&08&24', '2012@12^31' are all valid date formats. The only special punctuation is '.', which is treated as a decimal point to separate the integer and fractional parts. Date and time can be separated by T or a white space. For example, 2017-8-24 10:42:00 and 2017-8-24T10:42:00 represents the same date and time.

- 'YYYYMMDDHHMMSS' or 'YYMMDDHHMMSS': For example, '20170824104520' and '170824104520' are regarded as '2017-08-24 10:45:20'. However, if you provide a value out of range, such as '170824304520', it is not treated as a valid date. Note that incorrect formats such as YYYYMMDD HHMMSS, YYYYMMDD HH:MM:DD, or YYYY-MM-DD HHMMSS will fail to insert.
- YYYYMMDDHHMMSS or YYMMDDHHMMSS: Note that these formats have no single or double quotes, but a number. For example, 20170824104520 is interpreted as '2017-08-24 → 10:45:20'.

DATETIME or TIMESTAMP values can be followed by a fractional part, used to represent microseconds precision (6 digits). The fractional part should always be separated from the rest of the time by a decimal point ..

The year value containing only two digits is ambiguous. It is recommended to use the four-digit year format. TiDB interprets the two-digit year value according to the following rules:

- If the year value is in the range of 70-99, it is converted to 1970-1999.
- If the year value is in the range of 00-69, it is converted to 2000-2069.

For month or day values less than 10, '2017-8-4' is the same as '2017-08-04'. The same is true for Time. For example, '2017-08-24 1:2:3' is the same as '2017-08-24 → 01:02:03'.

When the date or time value is required, TiDB selects the specified format according to the length of the value:

- 6 digits: YYMMDD.
- 12 digits: YYMMDDHHMMSS.
- 8 digits: YYYYMMDD.
- 14 digits: YYYYMMDDHHMMSS.

TiDB supports the following formats for time values:

- 'D HH:MM:SS', or 'HH:MM:SS', 'HH:MM', 'D HH:MM', 'D HH', 'SS': D means days and the valid value range is 0-34.
- A number in HHMMSS format: For example, 231010 is interpreted as '23:10:10'.
- A number in any of SS, MMSS, and HHMMSS formats can be regarded as time.

The decimal point of the Time type is also ., with a precision of up to 6 digits after the decimal point.

See [MySQL date and time literals](#) for more details.

#### 11.5.1.2.4 Boolean Literals

The constants TRUE and FALSE are equal to 1 and 0 respectively, which are not case sensitive.

```
SELECT TRUE, true, tRuE, FALSE, FaLsE, false;
```

TRUE	true	tRuE	FALSE	FaLsE	false
1	1	1	0	0	0

1 row in set (0.00 sec)

#### 11.5.1.2.5 Hexadecimal literals

Hexadecimal literal values are written using X'val' or 0xval notation, where val contains hexadecimal digits. A leading 0x is case sensitive and cannot be written as 0X.

Legal hexadecimal literals:

```
X'ac12'
X'12AC'
x'ac12'
x'12AC'
0xac12
0x12AC
```

Illegal hexadecimal literals:

```
X'1z' (z is not a hexadecimal legal digit)
0X12AC (0X must be written as 0x)
```

Hexadecimal literals written using X'val' notation must contain an even number of digits. If the length of val is an odd number (for example, X'A' or X'11A'), to avoid the syntax error, pad the value with a leading zero:

```
mysql> select X'aff';
ERROR 1105 (HY000): line 0 column 13 near ""hex literal: invalid
      ↪ hexadecimal format, must even numbers, but 3 (total length 13)
mysql> select X'0aff';
+-----+
| X'0aff' |
+-----+
| 0x0aff  |
+-----+
1 row in set (0.00 sec)
```

By default, a hexadecimal literal is a binary string.

To convert a string or a number to a string in hexadecimal format, use the `HEX()` function:

```
mysql> SELECT HEX('TiDB');
+-----+
| HEX('TiDB') |
+-----+
| 54694442   |
+-----+
1 row in set (0.01 sec)

mysql> SELECT X'54694442';
+-----+
| X'54694442' |
+-----+
| TiDB        |
+-----+
1 row in set (0.00 sec)
```

#### 11.5.1.2.6 Bit-value literals

Bit-value literals are written using `b'val'` or `0bval` notation. The `val` is a binary value written using zeros and ones. A leading `0b` is case sensitive and cannot be written as `0B`.

Legal bit-value literals:

```
b'01'
B'01'
0b01
```

Illegal bit-value literals:

```
b'2' (2 is not a binary digit; it must be 0 or 1)
0B01 (0B must be written as 0b)
```

By default, a bit-value literal is a binary string.

Bit values are returned as binary values, which may not display well in the MySQL client. To convert a bit value to printable form, you can use a conversion function such as `BIN()` or `HEX()`.

```
CREATE TABLE t (b BIT(8));
INSERT INTO t SET b = b'00010011';
INSERT INTO t SET b = b'1110';
INSERT INTO t SET b = b'100101';

mysql> SELECT b+0, BIN(b), HEX(b) FROM t;
```

```
+-----+-----+-----+
| b+0 | BIN(b) | HEX(b) |
+-----+-----+-----+
| 19 | 10011 | 13   |
| 14 | 1110  | E    |
| 37 | 100101 | 25  |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

### 11.5.1.2.7 NULL Values

NULL means the data is empty, which is case-insensitive, and is synonymous with \N (case-sensitive).

#### Note:

NULL is not the same as 0, nor the empty string ''.

### 11.5.1.3 Schema Object Names

This document introduces schema object names in TiDB SQL statements.

Schema object names are used to name all schema objects in TiDB, including database, table, index, column, alias, and so on. You can quote these objects using identifiers in SQL statements.

You can use backticks to enclose the identifier. For example, `SELECT * FROM t` can also be written as `SELECT * FROM `t``. But if the identifier includes one or more special characters or is a reserved keyword, it must be enclosed in backticks to quote the schema object it represents.

```
SELECT * FROM `table` WHERE `table`.id = 20;
```

If you set `ANSI_QUOTES` in SQL MODE, TiDB will recognize the string enclosed in double quotation marks " as an identifier.

```
CREATE TABLE "test" (a varchar(10));
```

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
→ that corresponds to your TiDB version for the right syntax to use
→ line 1 column 19 near ""test" (a varchar(10))"
```

```
SET SESSION sql_mode='ANSI_QUOTES' ;
```

```
Query OK, 0 rows affected (0.000 sec)
```

```
CREATE TABLE "test" (a varchar(10));
```

```
Query OK, 0 rows affected (0.012 sec)
```

If you want to use the backtick character in the quoted identifier, repeat the backtick twice. For example, to create a table a`b:

```
CREATE TABLE `a``b` (a int);
```

In a SELECT statement, you can use an identifier or a string to specify an alias:

```
SELECT 1 AS `identifier`, 2 AS 'string';
```

identifier	string
1	2

```
1 row in set (0.00 sec)
```

For more information, see [MySQL Schema Object Names](#).

#### 11.5.1.3.1 Identifier qualifiers

Object names can be unqualified or qualified. For example, the following statement creates a table without a qualified name:

```
CREATE TABLE t (i int);
```

If you have not used the USE statement or the connection parameter to configure the database, the ERROR 1046 (3D000): No database selected error is displayed. At this time, you can specify the database qualified name:

```
CREATE TABLE test.t (i int);
```

White spaces can exist around .. table\_name.col\_name and table\_name . col\_name are equivalent.

To quote this identifier, use:

```
`table_name`.`col_name`
```

Instead of:

```
`table_name.col_name`
```

For more information, see [MySQL Identifier Qualifiers](#).

#### 11.5.1.4 Keywords

This article introduces the keywords in TiDB, the differences between reserved words and non-reserved words and summarizes all keywords for the query.

Keywords are words that have special meanings in SQL statements, such as `SELECT`, `UPDATE`, and `DELETE`. Some of them can be used as identifiers directly, which are called **non-reserved keywords**. Some of them require special treatment before being used as identifiers, which are called **reserved keywords**.

To use the reserved keywords as identifiers, you must enclose them in backticks `:

```
CREATE TABLE select (a INT);
```

```
ERROR 1105 (HY000): line 0 column 19 near " (a INT)" (total length 27)
```

```
CREATE TABLE `select` (a INT);
```

```
Query OK, 0 rows affected (0.09 sec)
```

The non-reserved keywords do not require backticks, such as `BEGIN` and `END`, which can be successfully used as identifiers in the following statement:

```
CREATE TABLE `select` (BEGIN int, END int);
```

```
Query OK, 0 rows affected (0.09 sec)
```

In the special case, the reserved keywords do not need backticks if they are used with the . delimiter:

```
CREATE TABLE test.select (BEGIN int, END int);
```

```
Query OK, 0 rows affected (0.08 sec)
```

The following list shows the keywords in TiDB. Reserved keywords are marked with (R). Reserved keywords for [Window Functions](#) are marked with (R-Window):

```
{< tabs-panel "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P"
 "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z" >}
```

A

- ACCOUNT
- ACTION
- ADD (R)
- ADMIN (R)
- ADVISE
- AFTER

- AGAINST
- AGO
- ALGORITHM
- ALL (R)
- ALTER (R)
- ALWAYS
- ANALYZE (R)
- AND (R)
- ANY
- AS (R)
- ASC (R)
- ASCII
- AUTO\_ID\_CACHE
- AUTO\_INCREMENT
- AUTO\_RANDOM
- AUTO\_RANDOM\_BASE
- AVG
- AVG\_ROW\_LENGTH

B

- BACKEND
- BACKUP
- BACKUPS
- BEGIN
- BETWEEN (R)
- BIGINT (R)
- BINARY (R)
- BINDING
- BINDINGS
- BINLOG
- BIT
- BLOB (R)
- BLOCK
- BOOL
- BOOLEAN
- BOTH (R)
- BTREE
- BUCKETS (R)
- BUILTINS (R)
- BY (R)
- BYTE

C

- CACHE
- CANCEL (R)
- CAPTURE
- CASCADE (R)
- CASCADED
- CASE (R)
- CHAIN
- CHANGE (R)
- CHAR (R)
- CHARACTER (R)
- CHARSET
- CHECK (R)
- CHECKPOINT
- CHECKSUM
- CIPHER
- CLEANUP
- CLIENT
- CMSKETCH (R)
- COALESCE
- COLLATE (R)
- COLLATION
- COLUMN (R)
- COLUMNS
- COLUMN\_FORMAT
- COMMENT
- COMMIT
- COMMITTED
- COMPACT
- COMPRESSED
- COMPRESSION
- CONCURRENCY
- CONFIG
- CONNECTION
- CONSISTENT
- CONSTRAINT (R)
- CONTEXT
- CONVERT (R)
- CPU
- CREATE (R)
- CROSS (R)
- CSV\_BACKSLASH\_ESCAPE
- CSV\_DELIMITER
- CSV\_HEADER
- CSV\_NOT\_NULL
- CSV\_NULL

- CSV\_SEPARATOR
- CSV\_TRIM\_LAST\_SEPARATORS
- CUME\_DIST (R-Window)
- CURRENT
- CURRENT\_DATE (R)
- CURRENT\_ROLE (R)
- CURRENT\_TIME (R)
- CURRENT\_TIMESTAMP (R)
- CURRENT\_USER (R)
- CYCLE

D

- DATA
- DATABASE (R)
- DATABASES (R)
- DATE
- DATETIME
- DAY
- DAY\_HOUR (R)
- DAY\_MICROSECOND (R)
- DAY\_MINUTE (R)
- DAY\_SECOND (R)
- DDL (R)
- DEALLOCATE
- DECIMAL (R)
- DEFAULT (R)
- DEFINER
- DELAYED (R)
- DELAY\_KEY\_WRITE
- DELETE (R)
- DENSE\_RANK (R-Window)
- DEPTH (R)
- DESC (R)
- DESCRIBE (R)
- DIRECTORY
- DISABLE
- DISCARD
- DISK
- DISTINCT (R)
- DISTINCTROW (R)
- DIV (R)
- DO
- DOUBLE (R)

- DRAINER (R)
- DROP (R)
- DUAL (R)
- DUPLICATE
- DYNAMIC

E

- ELSE (R)
- ENABLE
- ENCLOSED (R)
- ENCRYPTION
- END
- ENFORCED
- ENGINE
- ENGINES
- ENUM
- ERROR
- ERRORS
- ESCAPE
- ESCAPED (R)
- EVENT
- EVENTS
- EVOLVE
- EXCEPT (R)
- EXCHANGE
- EXCLUSIVE
- EXECUTE
- EXISTS (R)
- EXPANSION
- EXPIRE
- EXPLAIN (R)
- EXTENDED

F

- FALSE (R)
- FAULTS
- FIELDS
- FILE
- FIRST
- FIRST\_VALUE (R-Window)
- FIXED
- FLOAT (R)

- FLUSH
- FOLLOWING
- FOR (R)
- FORCE (R)
- FOREIGN (R)
- FORMAT
- FROM (R)
- FULL
- FULLTEXT (R)
- FUNCTION

G

- GENERAL
- GENERATED (R)
- GLOBAL
- GRANT (R)
- GRANTS
- GROUP (R)
- GROUPS (R-Window)

H

- HASH
- HAVING (R)
- HIGH\_PRIORITY (R)
- HISTORY
- HOSTS
- HOUR
- HOUR\_MICROSECOND (R)
- HOUR\_MINUTE (R)
- HOUR\_SECOND (R)

I

- IDENTIFIED
- IF (R)
- IGNORE (R)
- IMPORT
- IMPORTS
- IN (R)
- INCREMENT
- INCREMENTAL

- INDEX (R)
- INDEXES
- INFILE (R)
- INNER (R)
- INSERT (R)
- INSERT\_METHOD
- INSTANCE
- INT (R)
- INT1 (R)
- INT2 (R)
- INT3 (R)
- INT4 (R)
- INT8 (R)
- INTEGER (R)
- INTERVAL (R)
- INTO (R)
- INVISIBLE
- INVOKER
- IO
- IPC
- IS (R)
- ISOLATION
- ISSUER

J

- JOB (R)
- JOBS (R)
- JOIN (R)
- JSON

K

- KEY (R)
- KEYS (R)
- KEY\_BLOCK\_SIZE
- KILL (R)

L

- LABELS
- LAG (R-Window)
- LANGUAGE

- LAST
- LASTVAL
- LAST\_BACKUP
- LAST\_VALUE (R-Window)
- LEAD (R-Window)
- LEADING (R)
- LEFT (R)
- LESS
- LEVEL
- LIKE (R)
- LIMIT (R)
- LINEAR (R)
- LINES (R)
- LIST
- LOAD (R)
- LOCAL
- LOCALTIME (R)
- LOCALTIMESTAMP (R)
- LOCATION
- LOCK (R)
- LOGS
- LONG (R)
- LONGBLOB (R)
- LONGTEXT (R)
- LOW\_PRIORITY (R)

## M

- MASTER
- MATCH (R)
- MAXVALUE (R)
- MAX\_CONNECTIONS\_PER\_HOUR
- MAX\_IDNUM
- MAX\_MINUTES
- MAX\_QUERIES\_PER\_HOUR
- MAX\_ROWS
- MAX\_UPDATES\_PER\_HOUR
- MAX\_USER\_CONNECTIONS
- MB
- MEDIUMBLOB (R)
- MEDIUMINT (R)
- MEDIUMTEXT (R)
- MEMORY
- MERGE

- MICROSECOND
- MINUTE
- MINUTE\_MICROSECOND (R)
- MINUTE\_SECOND (R)
- MINVALUE
- MIN\_ROWS
- MOD (R)
- MODE
- MODIFY
- MONTH

N

- NAMES
- NATIONAL
- NATURAL (R)
- NCHAR
- NEVER
- NEXT
- NEXTVAL
- NO
- NOCACHE
- NOCYCLE
- NODEGROUP
- NODE\_ID (R)
- NODE\_STATE (R)
- NOMAXVALUE
- NOMINVALUE
- NONE
- NOT (R)
- NOWAIT
- NO\_WRITE\_TO\_BINLOG (R)
- NTH\_VALUE (R-Window)
- NTILE (R-Window)
- NULL (R)
- NULLS
- NUMERIC (R)
- NVARCHAR

O

- OFFSET
- ON (R)
- ONLINE

- ONLY
- ON\_DUPLICATE
- OPEN
- OPTIMISTIC (R)
- OPTIMIZE (R)
- OPTION (R)
- OPTIONALLY (R)
- OR (R)
- ORDER (R)
- OUTER (R)
- OUTFILE (R)
- OVER (R-Window)

P

- PACK\_KEYS
- PAGE
- PARSER
- PARTIAL
- PARTITION (R)
- PARTITIONING
- PARTITIONS
- PASSWORD
- PERCENT\_RANK (R-Window)
- PER\_DB
- PER\_TABLE
- PESSIMISTIC (R)
- PLUGINS
- PRECEDING
- PRECISION (R)
- PREPARE
- PRE\_SPLIT\_REGIONS
- PRIMARY (R)
- PRIVILEGES
- PROCEDURE (R)
- PROCESS
- PROCESSLIST
- PROFILE
- PROFILES
- PUMP (R)

Q

- QUARTER

- QUERIES
- QUERY
- QUICK

R

- RANGE (R)
- RANK (R-Window)
- RATE\_LIMIT
- READ (R)
- REAL (R)
- REBUILD
- RECOVER
- REDUNDANT
- REFERENCES (R)
- REGEXP (R)
- REGION (R)
- REGIONS (R)
- RELEASE (R)
- RELOAD
- REMOVE
- RENAME (R)
- REORGANIZE
- REPAIR
- REPEAT (R)
- REPEATABLE
- REPLACE (R)
- REPLICA
- REPLICATION
- REQUIRE (R)
- RESPECT
- RESTORE
- RESTORES
- RESTRICT (R)
- REVERSE
- REVOKE (R)
- RIGHT (R)
- RLIKE (R)
- ROLE
- ROLLBACK
- ROUTINE
- ROW (R)
- ROWS (R-Window)
- ROW\_COUNT

- ROW\_FORMAT
- ROW\_NUMBER (R-Window)
- RTREE

S

- SAMPLES (R)
- SECOND
- SECONDARY\_ENGINE
- SECONDARY\_LOAD
- SECONDARY\_UNLOAD
- SECOND\_MICROSECOND (R)
- SECURITY
- SELECT (R)
- SEND\_CREDENTIALS\_TO\_TIKV
- SEPARATOR
- SEQUENCE
- SERIAL
- SERIALIZABLE
- SESSION
- SET (R)
- SETVAL
- SHARD\_ROW\_ID\_BITS
- SHARE
- SHARED
- SHOW (R)
- SHUTDOWN
- SIGNED
- SIMPLE
- SKIP\_SCHEMA\_FILES
- SLAVE
- SLOW
- SMALLINT (R)
- SNAPSHOT
- SOME
- SOURCE
- SPATIAL (R)
- SPLIT (R)
- SQL (R)
- SQL\_BIG\_RESULT (R)
- SQL\_BUFFER\_RESULT
- SQL\_CACHE
- SQL\_CALC\_FOUND\_ROWS (R)
- SQL\_NO\_CACHE

- SQL\_SMALL\_RESULT (R)
- SQL\_TSI\_DAY
- SQL\_TSI\_HOUR
- SQL\_TSI\_MINUTE
- SQL\_TSI\_MONTH
- SQL\_TSI\_QUARTER
- SQL\_TSI\_SECOND
- SQL\_TSI\_WEEK
- SQL\_TSI\_YEAR
- SSL (R)
- START
- STARTING (R)
- STATS (R)
- STATS\_AUTO\_RECALC
- STATS\_BUCKETS (R)
- STATS\_HEALTHY (R)
- STATS\_HISTOGRAMS (R)
- STATS\_META (R)
- STATS\_PERSISTENT
- STATS\_SAMPLE\_PAGES
- STATUS
- STORAGE
- STORED (R)
- STRAIGHT\_JOIN (R)
- STRICT\_FORMAT
- SUBJECT
- SUBPARTITION
- SUBPARTITIONS
- SUPER
- SWAPS
- SWITCHES
- SYSTEM\_TIME

T

- TABLE (R)
- TABLES
- TABLESPACE
- TABLE\_CHECKSUM
- TEMPORARY
- TEMPTABLE
- TERMINATED (R)
- TEXT
- THAN

- THEN (R)
- TIDB (R)
- TIFLASH (R)
- TIKV\_IMPORTER
- TIME
- TIMESTAMP
- TINYBLOB (R)
- TINYINT (R)
- TINYTEXT (R)
- TO (R)
- TOPN (R)
- TRACE
- TRADITIONAL
- TRAILING (R)
- TRANSACTION
- TRIGGER (R)
- TRIGGERS
- TRUE (R)
- TRUNCATE
- TYPE

## U

- UNBOUNDED
- UNCOMMITTED
- UNDEFINED
- UNICODE
- UNION (R)
- UNIQUE (R)
- UNKNOWN
- UNLOCK (R)
- UNSIGNED (R)
- UPDATE (R)
- USAGE (R)
- USE (R)
- USER
- USING (R)
- UTC\_DATE (R)
- UTC\_TIME (R)
- UTC\_TIMESTAMP (R)

## V

- VALIDATION

- VALUE
- VALUES (R)
- VARBINARY (R)
- VARCHAR (R)
- VARCHARACTER (R)
- VARIABLES
- VARYING (R)
- VIEW
- VIRTUAL (R)
- VISIBLE

W

- WARNINGS
- WEEK
- WEIGHT\_STRING
- WHEN (R)
- WHERE (R)
- WIDTH (R)
- WINDOW (R-Window)
- WITH (R)
- WITHOUT
- WRITE (R)

X

- X509
- XOR (R)

Y

- YEAR
- YEAR\_MONTH (R)

Z

- ZEROFILL (R)

#### 11.5.1.5 User-Defined Variables

This document describes the concept of user-defined variables in TiDB and the methods to set and read the user-defined variables.

### Warning:

User-defined variables are still an experimental feature. It is **NOT** recommended that you use them in the production environment.

The format of the user-defined variables is `@var_name`. The characters that compose `var_name` can be any characters that can compose an identifier, including the numbers 0-9 → , the letters a-zA-Z, the underscore \_, the dollar sign \$, and the UTF-8 characters. In addition, it also includes the English period .. The user-defined variables are case-insensitive.

The user-defined variables are session-specific, which means a user variable defined by one client connection cannot be seen or used by other client connections.

#### 11.5.1.5.1 Set the user-defined variables

You can use the SET statement to set a user-defined variable, and the syntax is `SET ↪ @var_name = expr [, @var_name = expr] ...;`. For example:

```
SET @favorite_db = 'TiDB';
```

```
SET @a = 'a', @b = 'b', @c = 'c';
```

For the assignment operator, you can also use `:=`. For example:

```
SET @favorite_db := 'TiDB';
```

The content to the right of the assignment operator can be any valid expression. For example:

```
SET @c = @a + @b;
```

```
set @c = b'1000001' + b'1000001';
```

#### 11.5.1.5.2 Read the user-defined variables

To read a user-defined variable, you can use the SELECT statement to query:

```
SELECT @a1, @a2, @a3
```

	<code>@a1</code>	
	<code>@a2</code>	
	<code>@a3</code>	
+-----+-----+-----+		
	1	
	2	
	4	
+-----+-----+-----+		

You can also assign values in the SELECT statement:

```
SELECT @a1, @a2, @a3, @a4 := @a1+@a2+@a3;
```

@a1	@a2	@a3	@a4 := @a1+@a2+@a3
1	2	4	7

Before the variable `@a4` is modified or the connection is closed, its value is always 7.

If a hexadecimal literal or binary literal is used when setting the user-defined variable, TiDB will treat it as a binary string. If you want to set it to a number, you can manually add the `CAST` conversion, or use the numeric operator in the expression:

```
SET @v1 = b'1000001';
SET @v2 = b'1000001'+0;
SET @v3 = CAST(b'1000001' AS UNSIGNED);
```

```
SELECT @v1, @v2, @v3;
```

@v1	@v2	@v3
A	65	65

If you refer to a user-defined variable that has not been initialized, it has a value of `NULL` and a type of string.

```
SELECT @not_exist;
```

@not_exist
NULL

In addition to using the `SELECT` statement to read the user-defined variables, another common usage is the `PREPARE` statement. For example:

```
SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
PREPARE stmt FROM @s;
SET @a = 6;
SET @b = 8;
EXECUTE stmt USING @a, @b;
```

```
+-----+
| hypotenuse |
+-----+
|      10 |
+-----+
```

The contents of the user-defined variables are not recognized as identifiers in the SQL statements. For example:

```
SELECT * from t;
```

```
+---+
| a |
+---+
| 1 |
+---+
```

```
SET @col = ``a``;
SELECT @col FROM t;
```

```
+-----+
| @col |
+-----+
| `a` |
+-----+
```

#### 11.5.1.5.3 MySQL compatibility

Except for `SELECT ... INTO <variable>`, the syntax supported in MySQL and TiDB is identical.

For more information, see [User-Defined Variables in MySQL](#).

#### 11.5.1.6 Expression Syntax

An expression is a combination of one or more values, operators, or functions. In TiDB, expressions are mainly used in various clauses of the `SELECT` statement, including Group by clause, Where clause, Having clause, Join condition and window function. In addition, some DDL statements also use expressions, such as the setting of the default values, columns, and partition rules when creating tables.

The expressions can be divided into the following types:

- Identifier. For reference, see [Schema object names](#).

- Predicates, numeric values, strings, date expressions. The [Literal values](#) of these types are also expressions.
- Function calls and window functions. For reference, see [Functions and operators overview](#) and [Window functions](#)
- ParamMarker (?), system variables, user variables and CASE expressions.

The following rules are the expression syntax, which is based on the [parser.y](#) rules of TiDB parser. For the navigable version of the following syntax diagram, refer to [TiDB SQL Syntax Diagram](#).

```

Expression ::= 
  ( singleAtIdentifier assignmentEq | 'NOT' | Expression ( logOr | 'XOR' |
    ↪ logAnd ) ) Expression
| 'MATCH' '(' ColumnNameList ')' 'AGAINST' '(' BitExpr
  ↪ FulltextSearchModifierOpt ')'
| PredicateExpr ( IsOrNotOp 'NULL' | CompareOp ( ( singleAtIdentifier
  ↪ assignmentEq )? PredicateExpr | AnyOrAll SubSelect ) )* ( IsOrNotOp (
  ↪ trueKwd | falseKwd | 'UNKNOWN' ) )?

PredicateExpr ::= 
  BitExpr ( BetweenOrNotOp BitExpr 'AND' BitExpr )* ( InOrNotOp ( '('
    ↪ ExpressionList ')' | SubSelect ) | LikeOrNotOp SimpleExpr
    ↪ LikeEscapeOpt | RegexpOrNotOp SimpleExpr )?

BitExpr ::= 
  BitExpr ( ( '||' | '&' | '<<' | '>>' | '*' | '/' | '%' | 'DIV' | 'MOD' |
    ↪ '^' ) BitExpr | ( '+' | '-' ) ( BitExpr | "INTERVAL" Expression
    ↪ TimeUnit ) )
| SimpleExpr

SimpleExpr ::= 
  SimpleIdent ( ( '->' | '->>' ) stringLit )?
| FunctionCallKeyword
| FunctionCallNonKeyword
| FunctionCallGeneric
| SimpleExpr ( 'COLLATE' CollationName | pipes SimpleExpr )
| WindowFuncCall
| Literal
| paramMarker
| Variable
| SumExpr
| ( '!' | '~' | '-' | '+' | 'NOT' | 'BINARY' ) SimpleExpr
| 'EXISTS'? SubSelect

```

```
|  ( ( '(' ( ExpressionList ',' )? | 'ROW' '(' ExpressionList ',' )
  ↵ Expression | builtinCast '(' Expression 'AS' CastType | ( 'DEFAULT' |
  ↵ 'VALUES' ) '(' SimpleIdent | 'CONVERT' '(' Expression ( ',' CastType
  ↵ | 'USING' CharsetName ) ) ')'
| 'CASE' ExpressionOpt WhenClause+ ElseOpt 'END'
```

### 11.5.1.7 Comment Syntax

This document describes the comment syntax supported by TiDB.

TiDB supports three comment styles:

- Use # to comment a line:

<code>SELECT 1+1; # comments</code>
-------------------------------------

<pre>+-----+   1+1   +-----+      2   +-----+ 1 row in set (0.00 sec)</pre>
---

- Use -- to comment a line:

<code>SELECT 1+1; -- comments</code>
--------------------------------------

<pre>+-----+   1+1   +-----+      2   +-----+ 1 row in set (0.00 sec)</pre>
---

And this style requires at least one whitespace after --:

<code>SELECT 1+1--1;</code>
-----------------------------

<pre>+-----+   1+1--1   +-----+      3   +-----+ 1 row in set (0.01 sec)</pre>
--

- Use `/* */` to comment a block or multiple lines:

```
SELECT 1 /* this is an in-line comment */ + 1;
```

```
+-----+
| 1 + 1 |
+-----+
|      2 |
+-----+
1 row in set (0.01 sec)
```

```
SELECT 1+
/*
/*> this is a
/*> multiple-line comment
/*> */
1;
```

```
+-----+
| 1+
   1 |
+-----+
|      2 |
+-----+
1 row in set (0.001 sec)
```

#### 11.5.1.7.1 MySQL-compatible comment syntax

The same as MySQL, TiDB supports a variant of C comment style:

```
/*! Specific code */
```

or

```
/*!50110 Specific code */
```

In this style, TiDB runs the statements in the comment.

For example:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

In TiDB, you can also use another version:

```
SELECT STRAIGHT_JOIN col1 FROM table1,table2 WHERE ...
```

If the server version number is specified in the comment, for example, `/*!50110 → KEY_BLOCK_SIZE=1024 */`, in MySQL it means that the contents in this comment are processed only when the MySQL version is or higher than 5.1.10. But in TiDB, the MySQL version number does not work and all contents in the comment are processed.

#### 11.5.1.7.2 TiDB specific comment syntax

TiDB has its own comment syntax (that is, TiDB specific comment syntax), which can be divided into the following two types:

- `/*T! Specific code */`: This syntax can only be parsed and executed by TiDB, and be ignored in other databases.
- `/*T![feature_id] Specific code */`: This syntax is used to ensure compatibility between different versions of TiDB. TiDB can parse the SQL fragment in this comment only if it implements the corresponding feature of `feature_id` in the current version. For example, as the `AUTO_RANDOM` feature is introduced in v3.1.1, this version of TiDB can parse `/*T![auto_rand] auto_random */` into `auto_random`. Because the `AUTO_RANDOM` feature is not implemented in v3.0.0, the SQL statement fragment above is ignored. **Do not leave any space inside the `/*T![` characters.**

#### 11.5.1.7.3 Optimizer comment syntax

Another type of comment is specially treated as an optimizer hint:

```
SELECT /*+ hint */ FROM ...;
```

For details about the optimizer hints that TiDB supports, see [Optimizer hints](#).

#### Note

In MySQL client, the TiDB-specific comment syntax is treated as comments and cleared by default. In MySQL client before 5.7.7, hints are also seen as comments and are cleared by default. It is recommended to use the `--comments` option when you start the client. For example, `mysql -h 127.0.0.1 -P 4000 -uroot --comments`.

For more information, see [Comment Syntax](#).

### 11.5.2 SQL Statements

#### 11.5.2.1 ADD COLUMN

The `ALTER TABLE.. ADD COLUMN` statement adds a column to an existing table. This operation is online in TiDB, which means that neither reads or writes to the table are blocked by adding a column.

### 11.5.2.1.1 Synopsis

```

AlterTableStmt ::=

  'ALTER' IgnoreOptional 'TABLE' TableName ( AlterTableSpecListOpt
    ↪ AlterTablePartitionOpt | 'ANALYZE' 'PARTITION' PartitionNameList (
    ↪ 'INDEX' IndexNameList )? AnalyzeOptionListOpt )

AlterTableSpec ::=

  TableOptionList
| 'SET' 'TIFLASH' 'REPLICA' LengthNum LocationLabelList
| 'CONVERT' 'TO' CharsetKw ( CharsetName | 'DEFAULT' ) OptCollate
| 'ADD' ( ColumnKeywordOpt IfNotExists ( ColumnDef ColumnPosition | '(
  ↪ TableElementList ')' ) | Constraint | 'PARTITION' IfNotExists
  ↪ NoWriteToBinLogAliasOpt ( PartitionDefinitionListOpt | 'PARTITIONS'
  ↪ NUM ) )
| ( ( 'CHECK' | 'TRUNCATE' ) 'PARTITION' | ( 'OPTIMIZE' | 'REPAIR' | '
  ↪ REBUILD' ) 'PARTITION' NoWriteToBinLogAliasOpt )
  ↪ AllOrPartitionNameList
| 'COALESCE' 'PARTITION' NoWriteToBinLogAliasOpt NUM
| 'DROP' ( ColumnKeywordOpt IfExists ColumnName RestrictOrCascadeOpt | '
  ↪ PRIMARY' 'KEY' | 'PARTITION' IfExists PartitionNameList | (
  ↪ KeyOrIndex IfExists | 'CHECK' ) Identifier | 'FOREIGN' 'KEY' IfExists
  ↪ Symbol )
| 'EXCHANGE' 'PARTITION' Identifier 'WITH' 'TABLE' TableName
  ↪ WithValidationOpt
| ( 'IMPORT' | 'DISCARD' ) ( 'PARTITION' AllOrPartitionNameList )? '(
  ↪ TABLESPACE'
| 'REORGANIZE' 'PARTITION' NoWriteToBinLogAliasOpt
  ↪ ReorganizePartitionRuleOpt
| 'ORDER' 'BY' AlterOrderItem ( ',' AlterOrderItem )*
| ( 'DISABLE' | 'ENABLE' ) 'KEYS'
| ( 'MODIFY' ColumnKeywordOpt IfExists | 'CHANGE' ColumnKeywordOpt
  ↪ IfExists ColumnName ) ColumnDef ColumnPosition
| 'ALTER' ( ColumnKeywordOpt ColumnName ( 'SET' 'DEFAULT' ( SignedLiteral
  ↪ | '(' Expression ')' ) | 'DROP' 'DEFAULT' ) | 'CHECK' Identifier
  ↪ EnforcedOrNot | 'INDEX' Identifier IndexInvisible )
| 'RENAME' ( ( 'COLUMN' | KeyOrIndex ) Identifier 'TO' Identifier | ( 'TO'
  ↪ | '='? | 'AS' ) TableName )
| LockClause
| AlgorithmClause
| 'FORCE'
| ( 'WITH' | 'WITHOUT' ) 'VALIDATION'
| 'SECONDARY_LOAD'
| 'SECONDARY_UNLOAD'

```

```
ColumnDef ::=  
    ColumnName ( Type | 'SERIAL' ) ColumnOptionListOpt  
  
ColumnPosition ::=  
    ( 'FIRST' | 'AFTER' ColumnName )?
```

#### 11.5.2.1.2 Examples

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT);  
Query OK, 0 rows affected (0.11 sec)  
  
mysql> INSERT INTO t1 VALUES (NULL);  
Query OK, 1 row affected (0.02 sec)  
  
mysql> SELECT * FROM t1;  
+---+  
| id |  
+---+  
| 1 |  
+---+  
1 row in set (0.00 sec)  
  
mysql> ALTER TABLE t1 ADD COLUMN c1 INT NOT NULL;  
Query OK, 0 rows affected (0.28 sec)  
  
mysql> SELECT * FROM t1;  
+---+---+  
| id | c1 |  
+---+---+  
| 1 | 0 |  
+---+---+  
1 row in set (0.00 sec)  
  
mysql> ALTER TABLE t1 ADD c2 INT NOT NULL AFTER c1;  
Query OK, 0 rows affected (0.28 sec)  
  
mysql> SELECT * FROM t1;  
+---+---+---+  
| id | c1 | c2 |  
+---+---+---+  
| 1 | 0 | 0 |  
+---+---+---+  
1 row in set (0.00 sec)
```

### 11.5.2.1.3 MySQL compatibility

- Adding multiple columns at the same time in a statement is currently not supported.
- Adding a new column and setting it to the PRIMARY KEY is not supported.
- Adding a new column and setting it to AUTO\_INCREMENT is not supported.
- There are limitations on adding generated columns, refer to: [generated column limitations](#).

### 11.5.2.1.4 See also

- [ADD INDEX](#)
- [CREATE TABLE](#)

## 11.5.2.2 ADD INDEX

The ALTER TABLE.. ADD INDEX statement adds an index to an existing table. This operation is online in TiDB, which means that neither reads or writes to the table are blocked by adding an index.

### 11.5.2.2.1 Synopsis

```

AlterTableStmt ::=

  'ALTER' IgnoreOptional 'TABLE' TableName ( AlterTableSpecListOpt
    ↪ AlterTablePartitionOpt | 'ANALYZE' 'PARTITION' PartitionNameList (
    ↪ 'INDEX' IndexNameList )? AnalyzeOptionListOpt )

AlterTableSpec ::=

  TableOptionList
| 'SET' 'TIFLASH' 'REPLICA' LengthNum LocationLabelList
| 'CONVERT' 'TO' CharsetKw ( CharsetName | 'DEFAULT' ) OptCollate
| 'ADD' ( ColumnKeywordOpt IfNotExists ( ColumnDef ColumnPosition | '('
    ↪ TableElementList ')' ) | Constraint | 'PARTITION' IfNotExists
    ↪ NoWriteToBinLogAliasOpt ( PartitionDefinitionListOpt | 'PARTITIONS'
    ↪ NUM ) )
| ( ( 'CHECK' | 'TRUNCATE' ) 'PARTITION' | ( 'OPTIMIZE' | 'REPAIR' | '
    ↪ REBUILD' ) 'PARTITION' NoWriteToBinLogAliasOpt )
    ↪ AllOrPartitionNameList
| 'COALESCE' 'PARTITION' NoWriteToBinLogAliasOpt NUM
| 'DROP' ( ColumnKeywordOpt IfExists ColumnName RestrictOrCascadeOpt | '
    ↪ PRIMARY' 'KEY' | 'PARTITION' IfExists PartitionNameList | (
    ↪ KeyOrIndex IfExists | 'CHECK' ) Identifier | 'FOREIGN' 'KEY' IfExists
    ↪ Symbol )
| 'EXCHANGE' 'PARTITION' Identifier 'WITH' 'TABLE' TableName
    ↪ WithValidationOpt

```

```

| ( 'IMPORT' | 'DISCARD' ) ( 'PARTITION' AllOrPartitionNameList )? '
|   ↪ TABLESPACE'
| 'REORGANIZE' 'PARTITION' NoWriteToBinLogAliasOpt
|   ↪ ReorganizePartitionRuleOpt
| 'ORDER' 'BY' AlterOrderItem ( ',' AlterOrderItem )*
| ( 'DISABLE' | 'ENABLE' ) 'KEYS'
| ( 'MODIFY' ColumnKeywordOpt IfExists | 'CHANGE' ColumnKeywordOpt
|   ↪ IfExists ColumnName ) ColumnDef ColumnPosition
| 'ALTER' ( ColumnKeywordOpt ColumnName ( 'SET' 'DEFAULT' ( SignedLiteral
|   ↪ | '(' Expression ')' ) | 'DROP' 'DEFAULT' ) | 'CHECK' Identifier
|   ↪ EnforcedOrNot | 'INDEX' Identifier IndexInvisible )
| 'RENAME' ( ( 'COLUMN' | KeyOrIndex ) Identifier 'TO' Identifier | ( 'TO'
|   ↪ | '='? | 'AS' ) TableName )
| LockClause
| AlgorithmClause
| 'FORCE'
| ( 'WITH' | 'WITHOUT' ) 'VALIDATION'
| 'SECONDARY_LOAD'
| 'SECONDARY_UNLOAD'

Constraint ::= ConstraintKeywordOpt ConstraintElem

ConstraintKeywordOpt ::= ( 'CONSTRAINT' Symbol? )?

ConstraintElem ::= ( ( 'PRIMARY' 'KEY' | KeyOrIndex IfNotExists | 'UNIQUE' KeyOrIndexOpt )
|   ↪ IndexNameAndTypeOpt | 'FULLTEXT' KeyOrIndexOpt IndexName ) '('
|   ↪ IndexPartSpecificationList ')' IndexOptionList
| 'FOREIGN' 'KEY' IfNotExists IndexName '(' IndexPartSpecificationList ')'
|   ↪ ReferDef
| 'CHECK' '(' Expression ')' EnforcedOrNotOpt

IndexNameAndTypeOpt ::= IndexName ( 'USING' IndexTypeName )?
| Identifier 'TYPE' IndexTypeName

IndexPartSpecificationList ::= IndexPartSpecification ( ',' IndexPartSpecification )*
| Identifier 'TYPE' IndexTypeName

IndexPartSpecification ::= ( ColumnName OptFieldLen | '(' Expression ')' ) Order

IndexOptionList ::=

```

```

IndexOption*  
  

IndexOption ::=  

  'KEY_BLOCK_SIZE' '='? LengthNum  

| IndexType  

| 'WITH' 'PARSER' Identifier  

| 'COMMENT' stringLit  

| IndexInvisible  
  

KeyOrIndex ::=  

  'KEY'  

| 'INDEX'  
  

IndexKeyTypeOpt ::=  

  ('UNIQUE' | 'SPATIAL' | 'FULLTEXT')?  
  

IndexInvisible ::=  

  'VISIBLE'  

| 'INVISIBLE'  
  

IndexTypeName ::=  

  'BTREE'  

| 'HASH'  

| 'RTREE'

```

#### 11.5.2.2.2 Examples

```

mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT
      → NOT NULL);
Query OK, 0 rows affected (0.11 sec)

```

```

mysql> INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

```

```

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--+
  ↪-----+-----+-----+-----+
  ↪
  | id          | estRows | task      | access object | operator info
  ↪
+--+
  ↪-----+-----+-----+-----+
  ↪

```

```

| TableReader_7           | 10.00  | root      |                                | data:Selection_6
|   ↵                   |
| -Selection_6           | 10.00  | cop[tikv] |                                | eq(test.t1.c1,
|   ↵ 3)                 |
|   -TableFullScan_5     | 10000.00 | cop[tikv] | table:t1 | keep order:false
|   ↵ , stats:pseudo    |
+--+
|   ↵ -----+-----+-----+
|   ↵
3 rows in set (0.00 sec)

mysql> ALTER TABLE t1 ADD INDEX (c1);
Query OK, 0 rows affected (0.30 sec)

mysql> EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
+--+
| id          | estRows | task      | access object      | operator
|   ↵ info      |         |           |                  |
+--+
|   ↵ -----+-----+-----+
|   ↵
| IndexReader_6 | 0.01   | root      |                                | index:
|   ↵ IndexRangeScan_5 |
| -IndexRangeScan_5 | 0.01   | cop[tikv] | table:t1, index:c1(c1) | range
|   ↵ :[3,3], keep order:false, stats:pseudo |
+--+
|   ↵ -----+-----+-----+
|   ↵
2 rows in set (0.00 sec)

```

### 11.5.2.2.3 MySQL compatibility

- FULLTEXT, HASH and SPATIAL indexes are not supported.
- Descending indexes are not supported (similar to MySQL 5.7).
- Adding multiple indexes at the same time is currently not supported.
- Adding the primary key of the CLUSTERED type to a table is not supported. For more details about the primary key of the CLUSTERED type, refer to [clustered index](#).

### 11.5.2.2.4 See also

- [Index Selection](#)

- Wrong Index Solution
- CREATE INDEX
- DROP INDEX
- RENAME INDEX
- ALTER INDEX
- ADD COLUMN
- CREATE TABLE
- EXPLAIN

### 11.5.2.3 ADMIN

This statement is a TiDB extension syntax, used to view the status of TiDB and check the data of tables in TiDB.

#### 11.5.2.3.1 DDL related statement

Statement	Description
<code>ADMIN CANCEL DDL JOBS</code>	Cancels a currently running DDL job
<code>ADMIN CHECKSUM TABLE</code>	Calculates the CRC64 of all rows + index
<code>[ADMIN CHECK [TABLE INDEX]](#admin-check-[table index])</code>	
<code>[ADMIN SHOW DDL [JOBS QUERIES]](#admin-show-ddl-[jobs queries])</code>	

#### 11.5.2.3.2 ADMIN RELOAD statement

```
ADMIN RELOAD expr_pushdown_blacklist;
```

The above statement is used to reload the blocklist pushed down by the expression.

```
ADMIN RELOAD opt_rule_blacklist;
```

The above statement is used to reload the blocklist of logic optimization rules.

#### 11.5.2.3.3 ADMIN PLUGINS related statement

```
ADMIN PLUGINS ENABLE plugin_name [, plugin_name] ...;
```

The above statement is used to enable the `plugin_name` plugin.

```
ADMIN PLUGINS DISABLE plugin_name [, plugin_name] ...;
```

The above statement is used to disable the `plugin_name` plugin.

#### 11.5.2.3.4 ADMIN BINDINGS related statement

```
ADMIN FLUSH bindings;
```

The above statement is used to persist SQL Plan binding information.

```
ADMIN CAPTURE bindings;
```

The above statement can generate the binding of SQL Plan from the SELECT statement that occurs more than once.

```
ADMIN EVOLVE bindings;
```

After the automatic binding feature is enabled, the evolution of SQL Plan binding information is triggered every `bind-info-leave` (the default value is `3s`). The above statement is used to proactively trigger this evolution.

```
ADMIN RELOAD bindings;
```

The above statement is used to reload SQL Plan binding information.

#### 11.5.2.3.5 ADMIN REPAIR statement

To overwrite the metadata of the stored table in an untrusted way in extreme cases, use `ADMIN REPAIR TABLE`:

```
ADMIN REPAIR TABLE tbl_name CREATE TABLE STATEMENT;
```

Here “untrusted” means that you need to manually ensure that the metadata of the original table can be covered by the `CREATE TABLE STATEMENT` operation. To use this REPAIR statement, enable the `repair-mode` configuration item, and make sure that the tables to be repaired are listed in the `repair-table-list`.

#### 11.5.2.3.6 ADMIN SHOW SLOW statement

```
ADMIN SHOW SLOW RECENT N;
```

```
ADMIN SHOW SLOW TOP [INTERNAL | ALL] N;
```

For details, refer to [admin show slow statement](#)

#### 11.5.2.3.7 Synopsis

```
AdminStmt ::=  
  'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'  
    → 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'  
    → AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'  
    → TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '  
    → RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'  
    → TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'  
    → 'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
```

```

→ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
→ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
→ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
→ ' ) 'BINDINGS' )

```

#### 11.5.2.3.8 Examples

Run the following command to view the last 10 completed DDL jobs in the currently running DDL job queue. When NUM is not specified, only the last 10 completed DDL jobs is presented by default.

```
admin show ddl jobs;
```

JOB_ID	DB_NAME	TABLE_NAME	JOB_TYPE	SCHEMA_STATE	SCHEMA_ID
			TABLE_ID	ROW_COUNT	START_TIME
				STATE	END_TIME
→					
45	test	t1	add index	write reorganization	32
→ 37	0		2019-01-10 12:38:36.501 +0800 CST	running	
44	test	t1	add index	none	32
→ 37	0		2019-01-10 12:36:55.18 +0800 CST	2019-01-10	
→ 12:36:55.852 +0800 CST				rollback done	
43	test	t1	add index	public	32
→ 37	6		2019-01-10 12:35:13.66 +0800 CST	2019-01-10	
→ 12:35:14.925 +0800 CST				synced	
42	test	t1	drop index	none	32
→ 37	0		2019-01-10 12:34:35.204 +0800 CST	2019-01-10	
→ 12:34:36.958 +0800 CST				synced	
41	test	t1	add index	public	32
→ 37	0		2019-01-10 12:33:22.62 +0800 CST	2019-01-10	
→ 12:33:24.625 +0800 CST				synced	
40	test	t1	drop column	none	32
→ 37	0		2019-01-10 12:33:08.212 +0800 CST	2019-01-10	
→ 12:33:09.78 +0800 CST				synced	
39	test	t1	add column	public	32
→ 37	0		2019-01-10 12:32:55.42 +0800 CST	2019-01-10	
→ 12:32:56.24 +0800 CST				synced	
38	test	t1	create table	public	32
→ 37	0		2019-01-10 12:32:41.956 +0800 CST	2019-01-10	
→ 12:32:43.956 +0800 CST				synced	

36	test	drop table	none	32	
→ 34	0	2019-01-10 11:29:59.982 +0800 CST	2019-01-10		
→ 11:30:00.45 +0800 CST	synced				
35	test	create table	public	32	
→ 34	0	2019-01-10 11:29:40.741 +0800 CST	2019-01-10		
→ 11:29:41.682 +0800 CST	synced				
33	test	create schema	public	32	0
→	0	2019-01-10 11:29:22.813 +0800 CST	2019-01-10		
→ 11:29:23.954 +0800 CST	synced				
+-----+-----+-----+-----+-----+					
→					

Run the following command to view the last 5 completed DDL jobs in the currently running DDL job queue:

```
admin show ddl jobs 5;
```

JOB_ID	DB_NAME	TABLE_NAME	JOB_TYPE	SCHEMA_STATE	SCHEMA_ID
	TABLE_ID	ROW_COUNT	START_TIME		END_TIME
			STATE		
45	test	t1	add index	write reorganization	32
	37	0	2019-01-10 12:38:36.501 +0800 CST		
			running		
44	test	t1	add index	none	32
	37	0	2019-01-10 12:36:55.18 +0800 CST	2019-01-10	
			12:36:55.852 +0800 CST	rollback done	
43	test	t1	add index	public	32
	37	6	2019-01-10 12:35:13.66 +0800 CST	2019-01-10	
			12:35:14.925 +0800 CST	synced	
42	test	t1	drop index	none	32
	37	0	2019-01-10 12:34:35.204 +0800 CST	2019-01-10	
			12:34:36.958 +0800 CST	synced	
41	test	t1	add index	public	32
	37	0	2019-01-10 12:33:22.62 +0800 CST	2019-01-10	
			12:33:24.625 +0800 CST	synced	
40	test	t1	drop column	none	32
	37	0	2019-01-10 12:33:08.212 +0800 CST	2019-01-10	
			12:33:09.78 +0800 CST	synced	

Run the following command to view the uncompleted DDL jobs in the test database. The results include the DDL jobs that are running and the last 5 DDL jobs that are completed but failed.

```
admin show ddl jobs 5 where state!='synced' and db_name='test';
```

JOB_ID	DB_NAME	TABLE_NAME	JOB_TYPE	SCHEMA_STATE	SCHEMA_ID
TABLE_ID	ROW_COUNT	START_TIME		END_TIME	
		STATE			
45	test	t1	add index	write reorganization	32
37	0	2019-01-10 12:38:36.501 +0800 CST	running		
44	test	t1	add index	none	32
37	0	2019-01-10 12:36:55.18 +0800 CST	2019-01-10 12:36:55.852 +0800 CST	rollback done	

- **JOB\_ID**: each DDL operation corresponds to one DDL job. **JOB\_ID** is globally unique.
- **DB\_NAME**: the name of the database on which the DDL operations are performed.
- **TABLE\_NAME**: the name of the table on which the DDL operations are performed.
- **JOB\_TYPE**: the type of the DDL operations.
- **SCHEMA\_STATE**: the current state of the schema. If the **JOB\_TYPE** is `add index`, it is the state of the index; if the **JOB\_TYPE** is `add column`, it is the state of the column; if the **JOB\_TYPE** is `create table`, it is the state of the table. The common states include:
  - `none`: it indicates not existing. When the `drop` or `create` operation fails and rolls back, it usually becomes the `none` state.
  - `delete only`, `write only`, `delete reorganization`, `write reorganization`: these four states are intermediate states. These states are not visible in common operations, because the conversion from the intermediate states is so quick. You can see the `write reorganization` state only in `add index` operations, which means that the index data is being added.
  - `public`: it indicates existing and usable. When operations like `create table` and `add index/column` are finished, it usually becomes the `public` state, which means that the created table/column/index can be normally read and written now.
- **SCHEMA\_ID**: the ID of the database on which the DDL operations are performed.
- **TABLE\_ID**: the ID of the table on which the DDL operations are performed.
- **ROW\_COUNT**: the number of the data rows that have been added when running the `add index` operation.

- `START_TIME`: the start time of the DDL operations.
- `END_TIME`: the end time of the DDL operations.
- `STATE`: the state of the DDL operations. The common states include:
  - `none`: it indicates that the operation task has been put in the DDL job queue but has not been performed yet, because it is waiting for the previous tasks to complete. Another reason might be that it becomes the `none` state after running the drop operation, but it will soon be updated to the `synced` state, which means that all TiDB instances have been synced to this state.
  - `running`: it indicates that the operation is being performed.
  - `synced`: it indicates that the operation has been performed successfully and all TiDB instances have been synced to this state.
  - `rollback done`: it indicates that the operation has failed and has finished rolling back.
  - `rollingback`: it indicates that the operation has failed and is rolling back.
  - `cancelling`: it indicates that the operation is being cancelled. This state only occurs when you cancel DDL jobs using the `ADMIN CANCEL DDL JOBS` command.

#### 11.5.2.3.9 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

#### 11.5.2.4 ADMIN CANCEL DDL

The `ADMIN CANCEL DDL` statement allows you to cancel a running DDL job. The `job_id` can be found by running `ADMIN SHOW DDL JOBS`.

##### 11.5.2.4.1 Synopsis

```
AdminStmt ::=

  'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
    → 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
    → AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
    → TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | |
    → RECOVER 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
    → TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
    → 'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
    → ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
    → | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' |
    → TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
    → ' ) 'BINDINGS' )

NumList ::=

  Int64Num ( ',' Int64Num )*
```

#### 11.5.2.4.2 Examples

To cancel the currently running DDL jobs and return whether the corresponding jobs are successfully cancelled, use `ADMIN CANCEL DDL JOBS`:

```
ADMIN CANCEL DDL JOBS job_id [, job_id] ...;
```

If the operation fails to cancel the jobs, specific reasons are displayed.

**Note:**

- Only this operation can cancel DDL jobs. All other operations and environment changes (such as machine restart and cluster restart) cannot cancel these jobs.
- This operation can cancel multiple DDL jobs at the same time. You can get the ID of DDL jobs using the `ADMIN SHOW DDL JOBS` statement.
- If the jobs you want to cancel are finished, the cancellation operation fails.

#### 11.5.2.4.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

#### 11.5.2.4.4 See also

- [ADMIN SHOW DDL \[JOBS|QUERIES\]](#)

#### 11.5.2.5 ADMIN CHECKSUM TABLE

The `ADMIN CHECKSUM TABLE` statement calculates a CRC64 checksum for the data and indexes of a table. This statement is used by programs such as TiDB Lightning to ensure that import operations have completed successfully.

##### 11.5.2.5.1 Synopsis

```
AdminStmt ::=  
  'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'  
    ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'  
    ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'  
    ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '  
    ↪ RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'  
    ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'  
    ↪ 'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
```

```

→ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
→ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' |
→ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE'
→ ' ) 'BINDINGS' )

TableNameList ::=  

    TableName ( ',' TableName )*

```

### 11.5.2.5.2 Examples

Calculate the checksum for a table:

```

CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY auto_increment);
INSERT INTO t1 VALUES (1),(2),(3);
ADMIN CHECKSUM TABLE t1;

```

```
mysql> CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY auto_increment);
Query OK, 0 rows affected (0.11 sec)
```

```
mysql> INSERT INTO t1 VALUES (1),(2),(3);
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0
```

```
mysql> ADMIN CHECKSUM TABLE t1;
+-----+-----+-----+-----+
| Db_name | Table_name | Checksum_crc64_xor | Total_kvs | Total_bytes |
+-----+-----+-----+-----+
| test    | t1        | 10909174369497628533 |      3 |       75 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

### 11.5.2.5.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

### 11.5.2.6 ADMIN CHECK [TABLE|INDEX]

The `ADMIN CHECK [TABLE|INDEX]` statement checks for data consistency of tables and indexes.

#### 11.5.2.6.1 Synopsis

```
AdminStmt ::=
```

```
'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
    ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
    ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
    ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | '
    ↪ RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
    ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
    ↪ 'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
    ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
    ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
    ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
    ↪ ' ) 'BINDINGS' )
```

TableNameList ::=  
 TableName ( ',' TableName )\*

#### 11.5.2.6.2 Examples

To check the consistency of all the data and corresponding indexes in the `tbl_name` table, use `ADMIN CHECK TABLE`:

```
ADMIN CHECK TABLE tbl_name [, tbl_name] ...;
```

If the consistency check is passed, an empty result is returned. Otherwise, an error message is returned indicating that the data is inconsistent.

```
ADMIN CHECK INDEX tbl_name idx_name;
```

The above statement is used to check the consistency of the column data and index data corresponding to the `idx_name` index in the `tbl_name` table. If the consistency check is passed, an empty result is returned; otherwise, an error message is returned indicating that the data is inconsistent.

```
ADMIN CHECK INDEX tbl_name idx_name (lower_val, upper_val) [, (lower_val,
    ↪ upper_val)] ...;
```

The above statement is used to check the consistency of the column data and index data corresponding to the `idx_name` index in the `tbl_name` table, with the data range (to be checked) specified. If the consistency check is passed, an empty result is returned. Otherwise, an error message is returned indicating that the data is inconsistent.

#### 11.5.2.6.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

#### 11.5.2.6.4 See also

- `ADMIN REPAIR`

### 11.5.2.7 ADMIN SHOW DDL [JOBS|QUERIES]

The `ADMIN SHOW DDL [JOBS|QUERIES]` statement shows information about running and recently completed DDL jobs.

#### 11.5.2.7.1 Synopsis

```

AdminStmt ::=

  'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
    ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
    ↪ AdminShowSlow ) | 'CHECK' ( 'TABLE' TableNameList | 'INDEX'
    ↪ TableName Identifier ( HandleRange ( ',' HandleRange )* )? ) | |
    ↪ RECOVER 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
    ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
    ↪ 'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
    ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
    ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR' '
    ↪ TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
    ↪ ' ) 'BINDINGS' )

NumList ::=

  Int64Num ( ',' Int64Num )*

WhereClauseOptional ::=

  WhereClause?

```

#### 11.5.2.7.2 Examples

##### ADMIN SHOW DDL

To view the currently running DDL jobs, use `ADMIN SHOW DDL`:

```
ADMIN SHOW DDL;
```

```
mysql> ADMIN SHOW DDL;
+---+
→ -----+-----+
→
| SCHEMA_VER | OWNER_ID | OWNER_ADDRESS | RUNNING_JOBS
→ | SELF_ID | QUERY | |
+---+
→ -----+-----+
→
| 26 | 2d1982af-fa63-43ad-a3d5-73710683cc63 | 0.0.0.0:4000 | 2
→ d1982af-fa63-43ad-a3d5-73710683cc63 | |
```

```
+--+
→ -----
→
1 row in set (0.00 sec)
```

ADMIN SHOW DDL JOBS

To view all the results in the current DDL job queue (including tasks that are running and waiting to be run) and the last ten results in the completed DDL job queue, use ADMIN SHOW DDL JOBS:

```
ADMIN SHOW DDL JOBS;
```

```
mysql> ADMIN SHOW DDL JOBS;
+--+
→ -----
→
| JOB_ID | DB_NAME | TABLE_NAME      | JOB_TYPE      | SCHEMA_STATE      |
| SCHEMA_ID | TABLE_ID | ROW_COUNT | START_TIME      | END_TIME      |
| STATE |
```

```
+--+
→ -----
→
| 59 | test   | t1           | add index    | write reorganization |
| 1 | 55 | 88576 | 2020-08-17 07:51:58 | NULL          |
| running |
```

```
| 60 | test   | t2           | add index    | none          |
| 1 | 57 | 0 | 2020-08-17 07:51:59 | NULL          |
| none |
```

```
| 58 | test   | t2           | create table | public         |
| 1 | 57 | 0 | 2020-08-17 07:41:28 | 2020-08-17 |
| 07:41:28 | synced |
```

```
| 56 | test   | t1           | create table | public         |
| 1 | 55 | 0 | 2020-08-17 07:41:02 | 2020-08-17 |
| 07:41:02 | synced |
```

```
| 54 | test   | t1           | drop table   | none          |
| 1 | 50 | 0 | 2020-08-17 07:41:02 | 2020-08-17 |
| 07:41:02 | synced |
```

```
| 53 | test   | t1           | drop index   | none          |
| 1 | 50 | 0 | 2020-08-17 07:35:44 | 2020-08-17 |
| 07:35:44 | synced |
```

```
| 52 | test   | t1           | add index    | public         |
| 1 | 50 | 451010 | 2020-08-17 07:34:43 | 2020-08-17 |
| 07:35:16 | synced |
```

```
| 51 | test   | t1           | create table | public         |
| 1 | 50 | 0 | 2020-08-17 07:34:02 | 2020-08-17 |
| 07:34:02 | synced |
```

```

→ 07:34:02 | synced |
| 49 | test | t1 | drop table | none | 2020-08-17 07:34:02 | 2020-08-17
→ 1 | 47 | 0 | 2020-08-17 07:34:02 | 2020-08-17
→ 07:34:02 | synced |
| 48 | test | t1 | create table | public | 2020-08-17 07:33:37 | 2020-08-17
→ 1 | 47 | 0 | 2020-08-17 07:33:37 | 2020-08-17
→ 07:33:37 | synced |
| 46 | mysql | stats_extended | create table | public | 2020-08-17 06:42:38 | 2020-08-17
→ 3 | 45 | 0 | 2020-08-17 06:42:38 | 2020-08-17
→ 06:42:38 | synced |
| 44 | mysql | opt_rule_blacklist | create table | public | 2020-08-17 06:42:38 | 2020-08-17
→ 3 | 43 | 0 | 2020-08-17 06:42:38 | 2020-08-17
→ 06:42:38 | synced |
+--+
→ -----+-----+-----+-----+
→
12 rows in set (0.00 sec)

```

From the output above:

- Job 59 is currently in progress (**STATE** of **running**). The schema state is currently in **write reorganization**, but will switch to **public** once the task is completed to note that the change can be observed publicly by user sessions. The **end\_time** column is also **NULL** indicating that the completion time for the job is currently not known.
- Job 60 is an **add index** job, which is currently queued waiting for job 59 to complete. When job 59 completes, the **STATE** of job 60 will switch to **running**.
- For destructive changes such as dropping an index or dropping a table, the **SCHEMA\_STATE** will change to **none** when the job is complete. For additive changes, the **SCHEMA\_STATE** will change to **public**.

To limit the number of rows shown, specify a number and a where condition:

```
ADMIN SHOW DDL JOBS [NUM] [WHERE where_condition];
```

- **NUM**: to view the last **NUM** results in the completed DDL job queue. If not specified, **NUM** is by default 10.
- **WHERE**: to add filter conditions.

**ADMIN SHOW DDL JOB QUERIES**

To view the original SQL statements of the DDL job corresponding to **job\_id**, use **ADMIN SHOW DDL JOB QUERIES**:

```
ADMIN SHOW DDL JOBS;
ADMIN SHOW DDL JOB QUERIES 51;
```

```
mysql> ADMIN SHOW DDL JOB QUERIES 51;
+-----+
| QUERY |
+-----+
| CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY auto_increment) |
+-----+
1 row in set (0.02 sec)
```

You can only search the running DDL job corresponding to `job_id` within the last ten results in the DDL history job queue.

#### 11.5.2.7.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

#### 11.5.2.7.4 See also

- [ADMIN CANCEL DDL](#)

#### 11.5.2.8 ADMIN SHOW TELEMETRY

The `ADMIN SHOW TELEMETRY` statement shows the information that will be reported back to PingCAP as part of the [telemetry](#) feature.

##### 11.5.2.8.1 Synopsis

```
AdminStmt ::=

  'ADMIN' ( 'SHOW' ( 'DDL' ( 'JOBS' Int64Num? WhereClauseOptional | 'JOB'
    ↪ 'QUERIES' NumList )? | TableName 'NEXT_ROW_ID' | 'SLOW'
    ↪ AdminShowSlow | 'TELEMETRY' ) | 'CHECK' ( 'TABLE' TableNameList |
    ↪ 'INDEX' TableName Identifier ( HandleRange ( ',' HandleRange )* )?
    ↪ ) | 'RECOVER' 'INDEX' TableName Identifier | 'CLEANUP' ( 'INDEX'
    ↪ TableName Identifier | 'TABLE' 'LOCK' TableNameList ) | 'CHECKSUM'
    ↪ 'TABLE' TableNameList | 'CANCEL' 'DDL' 'JOBS' NumList | 'RELOAD'
    ↪ ( 'EXPR_PUSHDOWN_BLACKLIST' | 'OPT_RULE_BLACKLIST' | 'BINDINGS' )
    ↪ | 'PLUGINS' ( 'ENABLE' | 'DISABLE' ) PluginNameList | 'REPAIR'
    ↪ 'TABLE' TableName CreateTableStmt | ( 'FLUSH' | 'CAPTURE' | 'EVOLVE
    ↪ ' ) 'BINDINGS' )
```

### 11.5.2.8.2 Examples

```
ADMIN SHOW TELEMETRY\G
```

```
***** 1. row *****
TRACKING_ID: a1ba1d97-b940-4d5b-a9d5-ddb0f2ac29e7
LAST_STATUS: {
    "check_at": "2021-08-11T08:23:38+02:00",
    "is_error": false,
    "error_msg": "",
    "is_request_sent": true
}
DATA_PREVIEW: {
    "hardware": [
        {
            "instanceType": "tidb",
            "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
            "listenPort": "4000",
            "cpu": {
                "cache": "8192",
                "cpuFrequency": "2301.00MHz",
                "cpuLogicalCores": "8",
                "cpuPhysicalCores": "4"
            },
            "memory": {
                "capacity": "16410021888"
            },
            "disk": {
                "ebbca862689fa9fef7c55c3112e375c4ce575fe4": {
                    "deviceName": "ebbca862689fa9fef7c55c3112e375c4ce575fe4",
                    "free": "624438726656",
                    "freePercent": "0.61",
                    "fstype": "btrfs",
                    "opts": "bind,rw,relatime",
                    "path": "fb365c1216b59e1cf86950425867007a60f4435",
                    "total": "1022488477696",
                    "used": "397115568128",
                    "usedPercent": "0.39"
                },
                "nvme0n1p1": {
                    "deviceName": "nvme0n1p1",
                    "free": "582250496",
                    "freePercent": "0.93",
                    "fstype": "vfat",
                    "opts": "rw,relatime",
                }
            }
        }
    ]
}
```

```

    "path": "0fc8c8d71702d81a02e216fb6ef19f4dda4973df",
    "total": "627900416",
    "used": "45649920",
    "usedPercent": "0.07"
},
"nvme0n1p2": {
    "deviceName": "nvme0n1p2",
    "free": "701976576",
    "freePercent": "0.74",
    "fstype": "ext4",
    "opts": "rw,relatime",
    "path": "/boot",
    "total": "1023303680",
    "used": "250863616",
    "usedPercent": "0.26"
}
},
{
    "instanceType": "pd",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "2379",
    "cpu": {
        "cache": "8192",
        "cpuFrequency": "2301.00MHz",
        "cpuLogicalCores": "8",
        "cpuPhysicalCores": "4"
    },
    "memory": {
        "capacity": "16410021888"
    },
    "disk": {
        "ebbca862689fa9fef7c55c3112e375c4ce575fe4": {
            "deviceName": "ebbca862689fa9fef7c55c3112e375c4ce575fe4",
            "free": "624438726656",
            "freePercent": "0.61",
            "fstype": "btrfs",
            "opts": "bind,rw,relatime",
            "path": "fb365c1216b59e1cf86950425867007a60f4435",
            "total": "1022488477696",
            "used": "397115568128",
            "usedPercent": "0.39"
        },
        "nvme0n1p1": {
            "deviceName": "nvme0n1p1",

```

```

    "free": "582250496",
    "freePercent": "0.93",
    "fstype": "vfat",
    "opts": "rw,relatime",
    "path": "0fc8c8d71702d81a02e216fb6ef19f4dda4973df",
    "total": "627900416",
    "used": "45649920",
    "usedPercent": "0.07"
},
"nvmeOn1p2": {
    "deviceName": "nvmeOn1p2",
    "free": "701976576",
    "freePercent": "0.74",
    "fstype": "ext4",
    "opts": "rw,relatime",
    "path": "/boot",
    "total": "1023303680",
    "used": "250863616",
    "usedPercent": "0.26"
}
}
},
{
    "instanceType": "tikv",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "20160",
    "cpu": {
        "cpuFrequency": "3730MHz",
        "cpuLogicalCores": "8",
        "cpuPhysicalCores": "4",
        "cpuVendorId": "GenuineIntel",
        "l1CacheLineSize": "64",
        "l1CacheSize": "32768",
        "l2CacheLineSize": "64",
        "l2CacheSize": "262144",
        "l3CacheLineSize": "64",
        "l3CacheSize": "8388608"
},
    "memory": {
        "capacity": "16803861504"
},
    "disk": {
        "36e7dfacbb83843f83075d78aeb4cf850a4882a1": {
            "deviceName": "36e7dfacbb83843f83075d78aeb4cf850a4882a1",
            "free": "624438726656",

```

```

        "freePercent": "0.61",
        "fstype": "btrfs",
        "path": "fb365c1216b59e1cf86950425867007a60f4435",
        "total": "1022488477696",
        "used": "398049751040",
        "usedPercent": "0.39"
    },
    "nvme0n1p1": {
        "deviceName": "nvme0n1p1",
        "free": "582250496",
        "freePercent": "0.93",
        "fstype": "vfat",
        "path": "0fc8c8d71702d81a02e216fb6ef19f4dda4973df",
        "total": "627900416",
        "used": "45649920",
        "usedPercent": "0.07"
    },
    "nvme0n1p2": {
        "deviceName": "nvme0n1p2",
        "free": "701976576",
        "freePercent": "0.69",
        "fstype": "ext4",
        "path": "/boot",
        "total": "1023303680",
        "used": "321327104",
        "usedPercent": "0.31"
    }
},
{
    "instanceType": "tiflash",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "3930",
    "cpu": {
        "cpuFrequency": "3400MHz",
        "cpuLogicalCores": "8",
        "cpuPhysicalCores": "4",
        "l1CacheLineSize": "64",
        "l1CacheSize": "32768",
        "l2CacheLineSize": "64",
        "l2CacheSize": "262144",
        "l3CacheLineSize": "64",
        "l3CacheSize": "8388608"
    },
    "memory": {

```

```

        "capacity": "16410021888"
    },
    "disk": {
        "36e7dfacbb83843f83075d78aeb4cf850a4882a1": {
            "deviceName": "36e7dfacbb83843f83075d78aeb4cf850a4882a1",
            "free": "624438726656",
            "freePercent": "0.61",
            "fstype": "btrfs",
            "path": "fb365c1216b59e1fcfc86950425867007a60f4435",
            "total": "1022488477696",
            "used": "398049751040",
            "usedPercent": "0.39"
        }
    }
},
],
"instances": [
{
    "instanceType": "tidb",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "4000",
    "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "statusPort": "10080",
    "version": "5.1.1",
    "gitHash": "797bdd25310ed42f0791c8eccb78be8cce2f502",
    "startTime": "2021-08-11T08:23:38+02:00",
    "upTime": "22.210217487s"
},
{
    "instanceType": "pd",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "2379",
    "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "statusPort": "2379",
    "version": "5.1.1",
    "gitHash": "7cba1912b317a533e18b16ea2ba9a14ed2891129",
    "startTime": "2021-08-11T08:23:32+02:00",
    "upTime": "28.210220368s"
},
{
    "instanceType": "tikv",
    "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "listenPort": "20160",
    "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
    "statusPort": "20180",
}
]
}

```

```
"version": "5.1.1",
"gitHash": "4705d7c6e9c42d129d3309e05911ec6b08a25a38",
"startTime": "2021-08-11T08:23:33+02:00",
"upTime": "27.210221447s"
},
{
  "instanceType": "tiflash",
  "listenHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
  "listenPort": "3930",
  "statusHostHash": "4b84b15bff6ee5796152495a230e45e3d7e947d9",
  "statusPort": "20292",
  "version": "v5.1.1",
  "gitHash": "c8fabfb50fe28db17cc5118133a69be255c40efd",
  "startTime": "2021-08-11T08:23:40+02:00",
  "upTime": "20.210222452s"
}
],
"hostExtra": {
  "cpuFlags": [
    "fpu",
    "vme",
    "de",
    "pse",
    "tsc",
    "msr",
    "pae",
    "mce",
    "cx8",
    "apic",
    "sep",
    "mttr",
    "pge",
    "mca",
    "cmov",
    "pat",
    "pse36",
    "clflush",
    "dts",
    "acpi",
    "mmx",
    "fxsr",
    "sse",
    "sse2",
    "ss",
    "ht",
  ]
}
```

```
"tm",
"pbe",
"syscall",
"nx",
"pdpe1gb",
"rdtscp",
"lm",
"constant_tsc",
"art",
"arch_perfmon",
"pebs",
"bts",
"rep_good",
"nopl",
"xtopology",
"nonstop_tsc",
"cpuid",
"aperfmperf",
"pni",
"pclmulqdq",
"dtes64",
"monitor",
"ds_cpl",
"vmx",
"est",
"tm2",
:ssse3",
"sdbg",
"fma",
"cx16",
"xtpr",
"pdcm",
"pcid",
"sse4_1",
"sse4_2",
"x2apic",
"movbe",
"popcnt",
"tsc_deadline_timer",
"aes",
"xsav e",
"avx",
"f16c",
"rdrand",
"lahf_lm",
```

```
"abm",
"3dnowprefetch",
"cpuid_fault",
"epb",
"invpcid_single",
"ssbd",
"ibrs",
"ibpb",
"stibp",
"ibrs_enhanced",
"tpr_shadow",
"vnmi",
"flexpriority",
"ept",
"vpid",
"ept_ad",
"fsgsbase",
"tsc_adjust",
"sgx",
"bmi1",
"avx2",
"smepr",
"bmi2",
"erms",
"invpcid",
"mpx",
"rdseed",
"adx",
"smap",
"clflushopt",
"intel_pt",
"xsaveropt",
"xsaverc",
"xgetbv1",
"xsaves",
"dtherm",
"ida",
"arat",
"pln",
"pts",
"hwp",
"hwp_notify",
"hwp_act_window",
"hwp_epp",
"md_clear",
```

```

    "flush_l1d",
    "arch_capabilities"
],
"cpuModelName": "Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz",
"os": "linux",
"platform": "fedora",
"platformFamily": "fedora",
"platformVersion": "34",
"kernelVersion": "5.13.5-200.fc34.x86_64",
"kernelArch": "x86_64",
"virtualizationSystem": "kvm",
"virtualizationRole": "host"
},
"reportTimestamp": 1628663040,
"trackingId": "a1ba1d97-b940-4d5b-a9d5-ddb0f2ac29e7",
"featureUsage": {
  "txn": {
    "asyncCommitUsed": true,
    "onePCUsed": true,
    "txncCommitCounter": {
      "twoPC": 9,
      "asyncCommit": 0,
      "onePC": 0
    }
  },
  "clusterIndex": {},
  "temporaryTable": false,
  "cte": {
    "nonRecursiveCTEUsed": 0,
    "recursiveUsed": 0,
    "nonCTEUsed": 13
  }
},
>windowedStats": [],
"slowQueryStats": {
  "slowQueryBucket": {}
}
}
1 row in set (0.0259 sec)

```

### 11.5.2.8.3 MySQL compatibility

The ADMIN statement is a TiDB extension to MySQL syntax.

#### 11.5.2.8.4 See also

- [Telemetry](#)
- [tidb\\_enable\\_telemetry System Variable](#)

#### 11.5.2.9 ALTER DATABASE

`ALTER DATABASE` is used to specify or modify the default character set and collation of the current database. `ALTER SCHEMA` has the same effect as `ALTER DATABASE`.

##### 11.5.2.9.1 Synopsis

```
AlterDatabaseStmt ::=  
    'ALTER' 'DATABASE' DBName? DatabaseOptionList  
  
DatabaseOption ::=  
    DefaultKwdOpt ( CharsetKw '='? CharsetName | 'COLLATE' '='?  
        ↪ CollationName | 'ENCRYPTION' '='? EncryptionOpt )
```

##### 11.5.2.9.2 Examples

Modify the test database schema to use the `utf8mb4` character set:

```
ALTER DATABASE test DEFAULT CHARACTER SET = utf8mb4;
```

```
Query OK, 0 rows affected (0.00 sec)
```

Currently, TiDB only supports some character sets and collations. See [Character Set and Collation Support](#) for details.

##### 11.5.2.9.3 MySQL compatibility

This statement is understood to be fully compatible with MySQL. Any compatibility differences should be [reported via an issue](#) on GitHub.

#### 11.5.2.9.4 See also

- [CREATE DATABASE](#)
- [SHOW DATABASES](#)

#### 11.5.2.10 ALTER INDEX

The `ALTER INDEX` statement is used to modify the visibility of the index to `Visible` or `Invisible`. Invisible indexes are maintained by DML statements, but will not be used by the query optimizer. This is useful in scenarios where you want to double-check before removing an index permanently.

### 11.5.2.10.1 Synopsis

```

AlterTableStmt ::=

  'ALTER' IgnoreOptional 'TABLE' TableName ( AlterTableSpecListOpt
    ↪ AlterTablePartitionOpt | 'ANALYZE' 'PARTITION' PartitionNameList (
    ↪ 'INDEX' IndexNameList )? AnalyzeOptionListOpt )

AlterTableSpec ::=

  TableOptionList
| 'SET' 'TIFLASH' 'REPLICA' LengthNum LocationLabelList
| 'CONVERT' 'TO' CharsetKw ( CharsetName | 'DEFAULT' ) OptCollate
| 'ADD' ( ColumnKeywordOpt IfNotExists ( ColumnDef ColumnPosition | '('
    ↪ TableElementList ')' ) | Constraint | 'PARTITION' IfNotExists
    ↪ NoWriteToBinLogAliasOpt ( PartitionDefinitionListOpt | 'PARTITIONS'
    ↪ NUM ) )
| ( ( 'CHECK' | 'TRUNCATE' ) 'PARTITION' | ( 'OPTIMIZE' | 'REPAIR' | '
    ↪ REBUILD' ) 'PARTITION' NoWriteToBinLogAliasOpt )
    ↪ AllOrPartitionNameList
| 'COALESCE' 'PARTITION' NoWriteToBinLogAliasOpt NUM
| 'DROP' ( ColumnKeywordOpt IfExists ColumnName RestrictOrCascadeOpt | '
    ↪ PRIMARY' 'KEY' | 'PARTITION' IfExists PartitionNameList | (
    ↪ KeyOrIndex IfExists | 'CHECK' ) Identifier | 'FOREIGN' 'KEY' IfExists
    ↪ Symbol )
| 'EXCHANGE' 'PARTITION' Identifier 'WITH' 'TABLE' TableName
    ↪ WithValidationOpt
| ( 'IMPORT' | 'DISCARD' ) ( 'PARTITION' AllOrPartitionNameList )? ' '
    ↪ TABLESPACE'
| 'REORGANIZE' 'PARTITION' NoWriteToBinLogAliasOpt
    ↪ ReorganizePartitionRuleOpt
| 'ORDER' 'BY' AlterOrderItem ( ',' AlterOrderItem )*
| ( 'DISABLE' | 'ENABLE' ) 'KEYS'
| ( 'MODIFY' ColumnKeywordOpt IfExists | 'CHANGE' ColumnKeywordOpt
    ↪ IfExists ColumnName ) ColumnDef ColumnPosition
| 'ALTER' ( ColumnKeywordOpt ColumnName ( 'SET' 'DEFAULT' ( SignedLiteral
    ↪ | '(' Expression ')' ) | 'DROP' 'DEFAULT' ) | 'CHECK' Identifier
    ↪ EnforcedOrNot | 'INDEX' Identifier IndexInvisible )
| 'RENAME' ( ( 'COLUMN' | KeyOrIndex ) Identifier 'TO' Identifier | ( 'TO'
    ↪ | '='? | 'AS' ) TableName )
| LockClause
| AlgorithmClause
| 'FORCE'
| ( 'WITH' | 'WITHOUT' ) 'VALIDATION'
| 'SECONDARY_LOAD'
| 'SECONDARY_UNLOAD'

```

```
IndexInvisible ::=  
    'VISIBLE'  
| 'INVISIBLE'
```

#### 11.5.2.10.2 Examples

You can modify the visibility of an index using the `ALTER TABLE ... ALTER INDEX ...` statement.

```
CREATE TABLE t1 (c1 INT, UNIQUE(c1));  
ALTER TABLE t1 ALTER INDEX c1 INVISIBLE;
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
SHOW CREATE TABLE t1;
```

```
+--  
→ +-----+  
→  
| Table | Create Table  
|  
+--  
→ +-----+  
→  
| t1   | CREATE TABLE `t1` (  
  `c1` int(11) DEFAULT NULL,  
  UNIQUE KEY `c1` (`c1`) /*!80000 INVISIBLE */  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin |  
+--  
→ +-----+  
→  
1 row in set (0.00 sec)
```

The optimizer cannot use the `invisible index` of `c1`.

```
EXPLAIN SELECT c1 FROM t1 ORDER BY c1;
```

```
+--  
→ +-----+-----+-----+  
→  
| id          | estRows | task      | access object | operator info  
|  
+--  
→ +-----+-----+-----+
```

```

| Sort_4           | 10000.00 | root   |          | test.t1.c1:asc
|   ↵             |
| -TableReader_8  | 10000.00 | root   |          | data:
|   ↵ TableFullScan_7    |
|   -TableFullScan_7  | 10000.00 | cop[tikv] | table:t1 | keep order:false
|   ↵ , stats:pseudo |
+--+
|   ↵ -----+-----+-----+
|   ↵
3 rows in set (0.00 sec)

```

By comparison, c2 is a **visible index** and can be used by the optimizer.

```
EXPLAIN SELECT c2 FROM t1 ORDER BY c2;
```

```

+--+
| id           | estRows | task      | access object      | operator
|   ↵ info       |         |           |                   |
+--+
|   ↵ -----+-----+-----+
|   ↵
| IndexReader_13 | 10000.00 | root   |          | index:
|   ↵ IndexFullScan_12    |
| -IndexFullScan_12  | 10000.00 | cop[tikv] | table:t1, index:c2(c2) |
|   ↵ keep order:true, stats:pseudo |
+--+
|   ↵ -----+-----+-----+
|   ↵
2 rows in set (0.00 sec)

```

Even if you use the USE INDEX SQL hint to forcibly use indexes, the optimizer still cannot use invisible indexes; otherwise, an error is returned.

```
SELECT * FROM t1 USE INDEX(c1);
```

```
ERROR 1176 (42000): Key 'c1' doesn't exist in table 't1'
```

### Note:

“Invisible” here means invisible only to the optimizer. You can still modify or delete invisible indexes.

```
ALTER TABLE t1 DROP INDEX c1;
```

```
Query OK, 0 rows affected (0.02 sec)
```

#### 11.5.2.10.3 MySQL compatibility

- Invisible indexes in TiDB are modeled on the equivalent feature from MySQL 8.0.
- Similar to MySQL, TiDB does not permit PRIMARY KEY indexes to be made invisible.
- MySQL provides an optimizer switch `use_invisible_indexes=on` to make all invisible indexes *visible* again. This functionality is not available in TiDB.

#### 11.5.2.10.4 See also

- [CREATE TABLE](#)
- [CREATE INDEX](#)
- [ADD INDEX](#)
- [DROP INDEX](#)
- [RENAME INDEX](#)

### 11.5.2.11 ALTER INSTANCE

The `ALTER INSTANCE` statement is used to make changes to a single TiDB instance. Currently, TiDB only supports the `RELOAD TLS` clause.

#### 11.5.2.11.1 RELOAD TLS

You can execute the `ALTER INSTANCE RELOAD TLS` statement to reload the certificate (`ssl-cert`), the key (`ssl-key`), and the CA (`ssl-ca`) from the original configuration path.

The newly loaded certificate, key, and CA take effect on the connection that is established after the statement is successfully executed. The connection established before this statement execution is not affected.

When an error occurs during reloading, by default, this error message is returned and the previous key and certificate continue to be used. However, if you have added the optional `NO ROLLBACK ON ERROR`, when an error occurs during reloading, the error is not returned, and the subsequent requests are handled with the TLS security connection disabled.

#### 11.5.2.11.2 Syntax diagram

`AlterInstanceStmt:`

```

AlterInstanceStmt ::=

  'ALTER' 'INSTANCE' InstanceOption

InstanceOption ::=

  'RELOAD' 'TLS' ('NO' 'ROLLBACK' 'ON' 'ERROR')?

```

#### 11.5.2.11.3 Example

```
ALTER INSTANCE RELOAD TLS;
```

#### 11.5.2.11.4 MySQL compatibility

The `ALTER INSTANCE RELOAD TLS` statement only supports reloading from the original configuration path. It does not support dynamically modifying the loading path or dynamically enabling the TLS encrypted connection feature when TiDB is started. This feature is disabled by default when you restart TiDB.

#### 11.5.2.11.5 See also

[Enable TLS Between TiDB Clients and Servers.](#)

### 11.5.2.12 ALTER PLACEMENT POLICY

#### Warning:

Placement Rules in SQL is an experimental feature. The syntax might change before its GA, and there might also be bugs.

If you understand the risks, you can enable this experiment feature by executing `SET GLOBAL tidb_enable_alter_placement = 1;`.

`ALTER PLACEMENT POLICY` is used to modify existing placement policies that have previously been created. All the tables and partitions which use the placement policy will automatically be updated.

#### 11.5.2.12.1 Synopsis

```

AlterPolicyStmt ::=

  "ALTER" "PLACEMENT" "POLICY" IfExists PolicyName PlacementOptionList

PolicyName ::=


```

Identifier
<pre>PlacementOptionList ::=    DirectPlacementOption    PlacementOptionList DirectPlacementOption    PlacementOptionList ',' DirectPlacementOption</pre>
<pre>DirectPlacementOption ::=    "PRIMARY_REGION" EqOpt stringLit    "REGIONS" EqOpt stringLit    "FOLLOWERS" EqOpt LengthNum    "VOTERS" EqOpt LengthNum    "LEARNERS" EqOpt LengthNum    "SCHEDULE" EqOpt stringLit    "CONSTRAINTS" EqOpt stringLit    "LEADER_CONSTRAINTS" EqOpt stringLit    "FOLLOWER_CONSTRAINTS" EqOpt stringLit    "VOTER_CONSTRAINTS" EqOpt stringLit    "LEARNER_CONSTRAINTS" EqOpt stringLit</pre>

#### 11.5.2.12.2 Examples

**Note:**

To know which regions are available in your cluster, see [SHOW PLACEMENT](#) → [LABELS](#).

If you do not see any available regions, your TiKV installation might not have labels set correctly.

```
CREATE PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us
↪ -west-1";
ALTER PLACEMENT POLICY p1 PRIMARY_REGION="us-east-1" REGIONS="us-east-1,us-
↪ west-1,us-west-2" FOLLOWERS=4;
SHOW CREATE PLACEMENT POLICY p1\G
```

```
Query OK, 0 rows affected (0.08 sec)
```

```
Query OK, 0 rows affected (0.10 sec)
```

```
***** 1. row *****  

Policy: p1
```

```
Create Policy: CREATE PLACEMENT POLICY `p1` PRIMARY_REGION="us-east-1"
    ↪ REGIONS="us-east-1,us-west-1,us-west-2" FOLLOWERS=4
1 row in set (0.00 sec)
```

#### 11.5.2.12.3 MySQL compatibility

This statement is a TiDB extension to MySQL syntax.

#### 11.5.2.12.4 See also

- [Placement Rules in SQL](#)
- [SHOW PLACEMENT](#)
- [CREATE PLACEMENT POLICY](#)
- [DROP PLACEMENT POLICY](#)

### 11.5.2.13 ALTER TABLE

This statement modifies an existing table to conform to a new table structure. The statement `ALTER TABLE` can be used to:

- `ADD`, `DROP`, or `RENAME` indexes
- `ADD`, `DROP`, `MODIFY` or `CHANGE` columns

#### 11.5.2.13.1 Synopsis

```
AlterTableStmt ::=

  'ALTER' IgnoreOptional 'TABLE' TableName ( AlterTableSpecListOpt
    ↪ AlterTablePartitionOpt | 'ANALYZE' 'PARTITION' PartitionNameList (
    ↪ 'INDEX' IndexNameList )? AnalyzeOptionListOpt )

TableName ::=

  Identifier ('.' Identifier)?

AlterTableSpec ::=

  TableOptionList
| 'SET' 'TIFLASH' 'REPLICA' LengthNum LocationLabelList
| 'CONVERT' 'TO' CharsetKw ( CharsetName | 'DEFAULT' ) OptCollate
| 'ADD' ( ColumnKeywordOpt IfNotExists ( ColumnDef ColumnPosition | '('
    ↪ TableElementList ')' ) | Constraint | 'PARTITION' IfNotExists
    ↪ NoWriteToBinLogAliasOpt ( PartitionDefinitionListOpt | 'PARTITIONS'
    ↪ NUM ) )
| ( ( 'CHECK' | 'TRUNCATE' ) 'PARTITION' | ( 'OPTIMIZE' | 'REPAIR' |
    ↪ REBUILD' ) 'PARTITION' NoWriteToBinLogAliasOpt )
    ↪ AllOrPartitionNameList
```

```

|   'COALESCE' 'PARTITION' NoWriteToBinLogAliasOpt NUM
|   'DROP' ( ColumnKeywordOpt IfExists ColumnName RestrictOrCascadeOpt | '
|     ↪ PRIMARY' 'KEY' | 'PARTITION' IfExists PartitionNameList | (
|     ↪ KeyOrIndex IfExists | 'CHECK' ) Identifier | 'FOREIGN' 'KEY' IfExists
|     ↪ Symbol )
|   'EXCHANGE' 'PARTITION' Identifier 'WITH' 'TABLE' TableName
|     ↪ WithValidationOpt
|   ( 'IMPORT' | 'DISCARD' ) ( 'PARTITION' AllOrPartitionNameList )? '
|     ↪ TABLESPACE'
|   'REORGANIZE' 'PARTITION' NoWriteToBinLogAliasOpt
|     ↪ ReorganizePartitionRuleOpt
|   'ORDER' 'BY' AlterOrderItem ( ',' AlterOrderItem )*
|   ( 'DISABLE' | 'ENABLE' ) 'KEYS'
|   ( 'MODIFY' ColumnKeywordOpt IfExists | 'CHANGE' ColumnKeywordOpt
|     ↪ IfExists ColumnName ) ColumnDef ColumnPosition
|   'ALTER' ( ColumnKeywordOpt ColumnName ( 'SET' 'DEFAULT' ( SignedLiteral
|     ↪ | '(' Expression ')' ) | 'DROP' 'DEFAULT' ) | 'CHECK' Identifier
|     ↪ EnforcedOrNot | 'INDEX' Identifier IndexInvisible )
|   'RENAME' ( ( 'COLUMN' | KeyOrIndex ) Identifier 'TO' Identifier | ( 'TO'
|     ↪ | '='? | 'AS' ) TableName )
|   LockClause
|   AlgorithmClause
|   'FORCE'
|   ( 'WITH' | 'WITHOUT' ) 'VALIDATION'
|   'SECONDARY_LOAD'
|   'SECONDARY_UNLOAD'

```

### 11.5.2.13.2 Examples

Create a table with some initial data:

```

CREATE TABLE t1 (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT, c1 INT NOT
    ↪ NULL);
INSERT INTO t1 (c1) VALUES (1),(2),(3),(4),(5);

```

```
Query OK, 0 rows affected (0.11 sec)
```

```
Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0
```

The following query requires a full table scan because the column c1 is not indexed:

```
EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
```

```
+--+
| id          | estRows | task      | access object | operator info
+--+
| TableReader_7       | 10.00   | root      |               | data:Selection_6
| -Selection_6        | 10.00   | cop[tikv] |               | eq(test.t1.c1,
|   3)                |         |           |               |
|   -TableFullScan_5  | 10000.00 | cop[tikv] | table:t1 | keep order:false
|                   , stats:pseudo |
+--+
3 rows in set (0.00 sec)
```

The statement `ALTER TABLE .. ADD INDEX` can be used to add an index on the table t1. EXPLAIN confirms that the original query now uses an index range scan, which is more efficient:

```
ALTER TABLE t1 ADD INDEX (c1);
EXPLAIN SELECT * FROM t1 WHERE c1 = 3;
```

Query OK, 0 rows affected (0.30 sec)

```
+--+
| id          | estRows | task      | access object     | operator
| info        |         |           |                 |
+--+
| IndexReader_6        | 10.00   | root      |               | index:
|   IndexRangeScan_5    |         |           |               |
| -IndexRangeScan_5    | 10.00   | cop[tikv] | table:t1, index:c1(c1) | range
|   :[3,3], keep order:false, stats:pseudo |
+--+
2 rows in set (0.00 sec)
```

TiDB supports the ability to assert that DDL changes will use a particular ALTER algorithm. This is only an assertion, and does not change the actual algorithm which will be used to modify the table. It can be useful if you only want to permit instant DDL changes during the peak hours of your cluster:

```
ALTER TABLE t1 DROP INDEX c1, ALGORITHM=INSTANT;
```

```
Query OK, 0 rows affected (0.24 sec)
```

Using the `ALGORITHM=INSTANT` assertion on an operation that requires the `INPLACE` algorithm results in a statement error:

```
ALTER TABLE t1 ADD INDEX (c1), ALGORITHM=INSTANT;
```

```
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot
↪ alter table by INSTANT. Try ALGORITHM=INPLACE.
```

However, using the `ALGORITHM=COPY` assertion for an `INPLACE` operation generates a warning instead of an error. This is because TiDB interprets the assertion as *this algorithm or better*. This behavior difference is useful for MySQL compatibility because the algorithm TiDB uses might differ from MySQL:

```
ALTER TABLE t1 ADD INDEX (c1), ALGORITHM=COPY;
SHOW WARNINGS;
```

```
Query OK, 0 rows affected, 1 warning (0.25 sec)
```

Level	Code	Message
Error	1846	ALGORITHM=COPY is not supported. Reason: Cannot alter ↪ table by COPY. Try ALGORITHM=INPLACE.

```
+---+
↪ -----
↪ |
| Level | Code | Message
↪ |
↪ |
+---+
↪ -----
↪ |
| Error | 1846 | ALGORITHM=COPY is not supported. Reason: Cannot alter
↪ table by COPY. Try ALGORITHM=INPLACE. |
+---+
↪ -----
↪ |
1 row in set (0.00 sec)
```

### 11.5.2.13.3 MySQL compatibility

The following major restrictions apply to `ALTER TABLE` in TiDB:

- Making multiple changes in a single `ALTER TABLE` statement is currently not supported.
- Changes of the **Reorg-Data** types on primary key columns are not supported.
- Changes of column types on partitioned tables are not supported.
- Changes of column types on generated columns are not supported.
- Changes of some data types (for example, some TIME, Bit, Set, Enum, and JSON types) are not supported due to the compatibility issues of the `CAST` function's behavior between TiDB and MySQL.
- Spatial data types are not supported.

For further restrictions, see [MySQL Compatibility](#).

#### 11.5.2.13.4 See also

- [MySQL Compatibility](#)
- [ADD COLUMN](#)
- [DROP COLUMN](#)
- [ADD INDEX](#)
- [DROP INDEX](#)
- [RENAME INDEX](#)
- [ALTER INDEX](#)
- [CREATE TABLE](#)
- [DROP TABLE](#)
- [SHOW CREATE TABLE](#)

#### 11.5.2.14 ALTER USER

This statement changes an existing user inside the TiDB privilege system. In the MySQL privilege system, a user is the combination of a username and the host from which they are connecting from. Thus, it is possible to create a user '`'newuser2'@'192.168.1.1'`' who is only able to connect from the IP address `192.168.1.1`. It is also possible to have two users have the same user-portion, and different permissions as they login from different hosts.

##### 11.5.2.14.1 Synopsis

```

AlterUserStmt ::= 
  'ALTER' 'USER' IfExists (UserSpecList RequireClauseOpt ConnectionOptions
    ↪ PasswordOrLockOptions | 'USER' '(' ')' 'IDENTIFIED' 'BY'
    ↪ AuthString)

UserSpecList ::= 
  UserSpec ( ',' UserSpec )*

```

```

UserSpec ::= 
    Username AuthOption

Username ::= 
    StringName ('@' StringName | singleAtIdentifier)? | 'CURRENT_USER'
    ↪ OptionalBraces

AuthOption ::= 
    ( 'IDENTIFIED' ( 'BY' ( AuthString | 'PASSWORD' HashString ) | 'WITH'
        ↪ StringName ( 'BY' AuthString | 'AS' HashString )? ) )?

```

#### 11.5.2.14.2 Examples

```

mysql> CREATE USER 'newuser' IDENTIFIED BY 'newuserpassword';
Query OK, 1 row affected (0.01 sec)

mysql> SHOW CREATE USER 'newuser';
+--+
| CREATE USER for newuser@%
|   |
|   +-----+
|   |   |
|   |   | CREATE USER 'newuser'@'%' IDENTIFIED WITH 'mysql_native_password' AS '
|   |   |   ↪ *5806E04BBEE79E1899964C6A04D68BCA69B1A879' REQUIRE NONE PASSWORD
|   |   |   ↪ EXPIRE DEFAULT ACCOUNT UNLOCK |
|   |
+--+
|   |
|   +-----+
|   |   |
|   |   | CREATE USER for newuser@%
|   |   |
+--+
1 row in set (0.00 sec)

mysql> ALTER USER 'newuser' IDENTIFIED BY 'newnewpassword';
Query OK, 0 rows affected (0.02 sec)

mysql> SHOW CREATE USER 'newuser';
+--+
| CREATE USER for newuser@%
|   |
+--+

```

```

    ↵ |
+---+
    ↵ -----
    ↵
| CREATE USER 'newuser'@'%' IDENTIFIED WITH 'mysql_native_password' AS '*'
    ↵ FB8A1EA1353E8775CA836233E367FBDFCB37BE73' REQUIRE NONE PASSWORD
    ↵ EXPIRE DEFAULT ACCOUNT UNLOCK |
+---+
    ↵ -----
    ↵
1 row in set (0.00 sec)

```

#### 11.5.2.14.3 MySQL compatibility

- In MySQL this statement is used to change attributes such as to expire a password. This functionality is not yet supported by TiDB.

#### 11.5.2.14.4 See also

- [Security Compatibility with MySQL](#)
- [CREATE USER](#)
- [DROP USER](#)
- [SHOW CREATE USER](#)

#### 11.5.2.15 ANALYZE

This statement updates the statistics that TiDB builds on tables and indexes. It is recommended to run `ANALYZE` after performing a large batch update or import of records, or when you notice that query execution plans are sub-optimal.

TiDB will also automatically update its statistics over time as it discovers that they are inconsistent with its own estimates.

Currently, TiDB collects statistical information in two ways: full collection (implemented using the `ANALYZE TABLE` statement) and incremental collection (implemented using the `ANALYZE INCREMENTAL TABLE` statement). For detailed usage of these two statements, refer to [introduction to statistics](#)

#### 11.5.2.15.1 Synopsis

```
AnalyzeTableStmt ::=

  'ANALYZE' ( 'TABLE' ( TableNameList | TableName ( 'INDEX' IndexNameList?
    ↵   | 'COLUMNS' ColumnNameList | 'PARTITION' PartitionNameList ( '
    ↵   INDEX' IndexNameList? | 'COLUMNS' ColumnNameList )? )? )? | '
    ↵   'INCREMENTAL' 'TABLE' TableName ( 'PARTITION' PartitionNameList )?
    ↵   'INDEX' IndexNameList? ) AnalyzeOptionListOpt
```