TiDB on Kubernetes 用户文档

PingCAP Inc.

20250924

Table of Contents

1 TiDB on Kubernetes 文档 **5**

2 关于 TiDB Operator **5**

2.1 TiDB Operator 简介 5

2.1.1 TiDB Operator 与 TiDB 的版本兼容性 5

2.1.2 TiDB Operator v2 与 v1 的区别 5

2.1.3 使用 TiDB Operator 管理 TiDB 集群 5

3 在 Kubernetes 上快速上手 TiDB **6**

3.1 第 1 步：创建 Kubernetes 测试集群 7

3.2 第 2 步：部署 TiDB Operator 8

3.2.1 安装 TiDB Operator CRDs 8

3.2.2 安装 TiDB Operator 8

3.3 第 3 步：部署 TiDB 集群 8

3.4 第 4 步：连接 TiDB 集群 11

3.4.1 安装 `mysql` 命令行工具 11

3.4.2 转发 TiDB 服务 4000 端口 11

3.4.3 连接 TiDB 服务 12

4 部署 **13**

4.1 在 Kubernetes 上部署 TiDB Operator 13
4.1.1 准备环境 13
4.1.2 部署 Kubernetes 集群 13
4.1.3 部署 TiDB Operator CRD 14
4.1.4 部署 TiDB Operator 14
4.2 在 Kubernetes 上部署 TiDB 集群 15
4.2.1 前提条件 15
4.2.2 配置 TiDB 集群 15
4.2.3 部署 TiDB 集群 19
4.3 访问 Kubernetes 上的 TiDB 集群 21
4.3.1 ClusterIP 21
4.3.2 NodePort 23
4.3.3 LoadBalancer 23

5 监控与告警 24
5.1 TiDB 集群的监控与告警 24
5.1.1 TiDB 集群的监控 24
5.1.2 告警配置 31
5.1.3 使用 Grafana 查看多集群监控 31
5.2 Kubernetes 可观测性：监控、告警与日志收集 31
5.2.1 监控 31
5.2.2 告警 32
5.2.3 日志收集 32

6 集群配置 33
6.1 组件配置 33
6.1.1 配置 TiDB 配置参数 33
6.1.2 配置 TiKV 配置参数 34
6.1.3 配置 PD 配置参数 34
6.1.4 配置 TiProxy 配置参数 36
6.1.5 配置 TiFlash 配置参数 37
6.1.6 配置 TiCDC 启动参数 37

6.2 存储卷配置 38
6.2.1 概述 38
6.2.2 组件特定的存储卷配置 38
6.2.3 修改存储卷 40
6.2.4 常见问题 43
6.3 自定义 Kubernetes 原生资源的配置 44
6.3.1 支持的资源类型 44
6.3.2 使用方法 44
6.3.3 注意事项 45
6.3.4 示例场景 46

7 运维管理 48
7.1 安全 48
7.1.1 为 MySQL 客户端开启 TLS 48
7.1.2 为 TiDB 组件间开启 TLS 57
7.1.3 以非 root 用户运行容器 71
7.1.4 更新和替换 TLS 证书 72
7.2 手动扩缩容 Kubernetes 上的 TiDB 集群 77
7.2.1 水平扩缩容 77
7.2.2 垂直扩缩容 78
7.2.3 扩缩容故障诊断 79
7.3 升级 79
7.3.1 升级 TiDB Operator 79
7.3.2 升级 Kubernetes 上的 TiDB 集群 81
7.4 备份与恢复 82
7.4.1 备份与恢复简介 82
7.4.2 备份与恢复 CR 介绍 85
7.4.3 远程存储访问授权 94
7.4.4 使用 Amazon S3 兼容的存储 101
7.4.5 使用 Google Cloud Storage 129
7.4.6 使用 Azure Blob Storage 152

7.5 运维 175
7.5.1 查看日志 175
7.5.2 暂停同步 Kubernetes 上的 TiDB 集群 175
7.5.3 挂起和恢复 Kubernetes 上的 TiDB 集群 177
7.5.4 重启 Kubernetes 上的 TiDB 集群 179
7.5.5 销毁 Kubernetes 上的 TiDB 集群 180

8 故障诊断 180
8.1 Kubernetes 上的 TiDB 常见部署错误 180
8.1.1 Pod 未正常创建 180
8.1.2 Pod 处于 Pending 状态 181
8.1.3 Pod 处于 CrashLoopBackOff 状态 181
8.2 Kubernetes 上的 TiDB 集群常见异常 182
8.2.1 TiDB 长连接被异常中断 182

9 参考 183
9.1 架构 183
9.1.1 TiDB Operator 架构 183
9.2 TiDB Operator v2 和 v1 的对比 186
9.2.1 TiDB Operator v2 的核心变更 186
9.2.2 TiDB Operator v2 暂不支持的组件和功能 187
9.3 工具 189
9.3.1 Kubernetes 上的 TiDB 工具指南 189

10 版本发布历史 190
10.1 v2.0 190
10.1.1 TiDB Operator 2.0.0-beta.0 Release Notes 190

1 TiDB on Kubernetes 文档

2 关于 TiDB Operator

2.1 TiDB Operator 简介

TiDB Operator 是 Kubernetes 上的 TiDB 集群自动运维系统，提供包括部署、升级、扩缩容、备份恢复、配置变更的 TiDB 全生命周期管理。借助 TiDB Operator，TiDB 可以无缝运行在公有云或自托管的 Kubernetes 集群上。

2.1.1 TiDB Operator 与 TiDB 的版本兼容性

TiDB Operator 与适用的 TiDB 版本的对应关系如下：

TiDB 版本	适用的 TiDB Operator 版本
dev	dev
TiDB >= 8.0	2.0，1.6（推荐），1.5
7.1 <= TiDB < 8.0	1.5（推荐），1.4
6.5 <= TiDB < 7.1	1.5，1.4（推荐），1.3
5.4 <= TiDB < 6.5	1.4，1.3（推荐）
5.1 <= TiDB < 5.4	1.4，1.3（推荐），1.2（停止维护）
3.0 <= TiDB < 5.1	1.4，1.3（推荐），1.2（停止维护），1.1（停止维护）
2.1 <= TiDB < v3.0	1.0（停止维护）

2.1.2 TiDB Operator v2 与 v1 的区别

随着 TiDB 和 Kubernetes 生态的快速发展，TiDB Operator 发布了与 v1 不兼容的 v2 版本。关于 v2 与 v1 的详细差异，请参考TiDB Operator v2 与 v1 版本对比。

2.1.3 使用 TiDB Operator 管理 TiDB 集群

在 Kubernetes 环境中，TiDB Operator 可用于高效部署和管理 TiDB 集群。你可以根据不同需求选择以下部署方式：

- 如需在测试环境中快速部署 TiDB Operator 并搭建一个 TiDB 集群，请参考快速开始。
- 如需自定义部署 TiDB Operator，请参阅部署 TiDB Operator。

在任何环境上部署前，都可以参考下面的文档来自定义 TiDB 配置：

- 存储卷配置

- 自定义 Pod

部署完成后，你可以参考下面的文档进行 Kubernetes 上 TiDB 集群的使用和运维：

- 部署 TiDB 集群
- 访问 TiDB 集群
- TiDB 集群扩缩容
- 查看 TiDB 日志

当集群出现问题需要进行诊断时，你可以：

- 查阅 Kubernetes 上的 TiDB FAQ 寻找是否存在现成的解决办法；
- 参考 Kubernetes 上的 TiDB 故障诊断解决故障。

在 Kubernetes 上，TiDB 的部分生态工具的使用方法也有所不同，你可以参考Kubernetes 上的 TiDB 相关工具使用指南来了解 TiDB 生态工具在 Kubernetes 上的使用方法。

最后，当 TiDB Operator 发布新版本时，你可以参考升级 TiDB Operator 进行版本更新。

3 在 Kubernetes 上快速上手 TiDB

本文档介绍了如何创建一个简单的 Kubernetes 集群，部署 TiDB Operator，并使用 TiDB Operator 部署 TiDB 集群。

> **警告：**
>
> 本文中的部署说明仅用于测试目的，不要直接用于生产环境。如果要在生产环境部署，请参阅在 Kubernetes 上部署 TiDB 集群。

部署的基本步骤如下：

1. 创建 Kubernetes 测试集群
2. 部署 TiDB Operator
3. 部署 TiDB 集群
4. 连接 TiDB 集群

3.1 第 1 步：创建 Kubernetes 测试集群

本节介绍如何使用 kind 创建一个 Kubernetes 测试集群。你也可以参考 Kubernetes 官方文档，选择其他方法部署 Kubernetes 集群。

kind 可以使用容器作为集群节点运行本地 Kubernetes 集群。请参阅 kind 官方文档完成安装。

以下以 kind 0.24.0 版本为例：

```
kind create cluster --name tidb-operator
```

点击查看期望输出

```
create cluster with image kindest/node:v1.31.0@sha256:53
    ↪ df588e04085fd41ae12de0c3fe4c72f7013bba32a20e7325357a1ac94ba865
Creating cluster "tidb-operator" ...
   Ensuring node image (kindest/node:v1.31.0) 
   Preparing nodes     
   Writing configuration 
   Starting control-plane 
   Installing CNI 
   Installing StorageClass 
   Joining worker nodes 
Set kubectl context to "kind-tidb-operator"
You can now use your cluster with:

kubectl cluster-info --context kind-tidb-operator

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s
    ↪ .io/#community 
```

检查集群是否创建成功：

```
kubectl cluster-info --context kind-tidb-operator
```

点击查看期望输出

```
Kubernetes master is running at https://127.0.0.1:51026
KubeDNS is running at https://127.0.0.1:51026/api/v1/namespaces/kube-system/
    ↪ services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info
    ↪ dump'.
```

Kubernetes 集群部署完成，现在就可以开始部署 TiDB Operator 了！

3.2 第 2 步：部署 TiDB Operator

部署 TiDB Operator 的过程分为两步：

1. 安装 TiDB Operator CRDs
2. 安装 TiDB Operator

3.2.1 安装 TiDB Operator CRDs

TiDB Operator 包含许多实现 TiDB 集群不同组件的自定义资源类型 (CRD)。执行以下命令安装 CRD 到集群中：

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.crds.yaml --server-side
```

3.2.2 安装 TiDB Operator

执行以下命令安装 TiDB Operator 到集群中：

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.yaml --server-side
```

检查 TiDB Operator 组件是否正常运行起来：

```
kubectl get pods --namespace tidb-admin
```

点击查看期望输出

```
NAME                            READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m22s
```

当所有的 pods 都处于 Running 状态时，继续下一步。

3.3 第 3 步：部署 TiDB 集群

按照以下步骤部署 TiDB 集群：

1. 创建命名空间 Namespace：

> 注意：
> 暂不支持跨 Namespace 引用 `Cluster`。请确保所有组件部署在同一个 Kubernetes Namespace 中。

```
kubectl create namespace db
```

2. 部署 TiDB 集群：

   方法一：使用以下命令创建一个包含 PD、TiKV 和 TiDB 组件的 TiDB 集群

   创建 Cluster：

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: basic
  namespace: db
spec: {}
```

```
kubectl apply -f cluster.yaml --server-side
```

   创建 PD 组件：

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 1
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 20Gi
```

```
kubectl apply -f pd.yaml --server-side
```

   创建 TiKV 组件：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
```

```
  name: tikv
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 1
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

```
kubectl apply -f tikv.yaml --server-side
```

创建 TiDB 组件：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: tidb
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 1
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
```

```
kubectl apply -f tidb.yaml --server-side
```

方法二：将以上 YAML 文件保存到本地目录中，并使用以下命令一次性部署 TiDB 集群

```
kubectl apply -f ./<directory> --server-side
```

3. **查看 Pod 状态：**

```
watch kubectl get pods -n db
```

点击查看期望输出

```
NAME              READY   STATUS    RESTARTS  AGE
pd-pd-68t96d      1/1     Running   0         2m
tidb-tidb-coqwpi  1/1     Running   0         2m
tikv-tikv-sdoxy4  1/1     Running   0         2m
```

所有组件的 Pod 都启动后，每种类型组件（pd、tikv 和 tidb）都会处于 Running 状态。此时，你可以按 Ctrl+C 返回命令行，然后进行下一步。

3.4 第 4 步：连接 TiDB 集群

由于 TiDB 支持 MySQL 传输协议及其绝大多数的语法，因此你可以直接使用 mysql 命令行工具连接 TiDB 进行操作。以下说明连接 TiDB 集群的步骤。

3.4.1 安装 mysql 命令行工具

要连接到 TiDB，你需要在使用 kubectl 的主机上安装与 MySQL 兼容的命令行客户端。可以安装 MySQL Server、MariaDB Server 和 Percona Server 的 MySQL 可执行文件，也可以从操作系统软件仓库中安装。

3.4.2 转发 TiDB 服务 4000 端口

本步骤将端口从本地主机转发到 Kubernetes 中的 TiDB **Service**。

首先，获取 db 命名空间中的服务列表：

```
kubectl get svc -n db
```

点击查看期望输出

```
NAME           TYPE       CLUSTER-IP     EXTERNAL-IP PORT(S)             AGE
pd-pd          ClusterIP  10.96.229.12   <none>      2379/TCP,2380/TCP   3m
pd-pd-peer     ClusterIP  None           <none>      2379/TCP,2380/TCP   3m
tidb-tidb      ClusterIP  10.96.174.237  <none>      4000/TCP,10080/TCP  3m
tidb-tidb-peer ClusterIP  None           <none>      10080/TCP           3m
tikv-tikv-peer ClusterIP  None           <none>      20160/TCP,20180/TCP 3m
```

这个例子中，TiDB **Service** 是 **tidb-tidb**。

然后，使用以下命令转发本地端口到集群：

```
kubectl port-forward -n db svc/tidb-tidb 14000:4000 > pf14000.out &
```

如果端口 14000 已经被占用，可以更换一个空闲端口。命令会在后台运行，并将输出转发到文件 pf14000.out。所以，你可以继续在当前 shell 会话中执行命令。

3.4.3 连接 TiDB 服务

```
mysql --comments -h 127.0.0.1 -P 14000 -u root
```

点击查看期望输出

```
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 178505
Server version: 8.0.11-TiDB-v8.5.2 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 8.0 compatible

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
    ↪ statement.

MySQL [(none)]>
```

以下是一些可以用来验证集群功能的命令。

创建 hello_world 表

```
mysql> use test;
mysql> create table hello_world (id int unsigned not null auto_increment
    ↪ primary key, v varchar(32));
Query OK, 0 rows affected (0.17 sec)

mysql> select * from information_schema.tikv_region_status where db_name=
    ↪ database() and table_name='hello_world'\G
*************************** 1. row ***************************
              REGION_ID: 18
              START_KEY: 7480000000000000FF6800000000000000F8
                END_KEY: 748000FFFFFFFFFFFF900000000000000F8
               TABLE_ID: 104
                DB_NAME: test
             TABLE_NAME: hello_world
               IS_INDEX: 0
               INDEX_ID: NULL
             INDEX_NAME: NULL
           IS_PARTITION: 0
           PARTITION_ID: NULL
         PARTITION_NAME: NULL
         EPOCH_CONF_VER: 5
          EPOCH_VERSION: 57
          WRITTEN_BYTES: 0
             READ_BYTES: 0
       APPROXIMATE_SIZE: 1
```

```
        APPROXIMATE_KEYS: 0
  REPLICATIONSTATUS_STATE: NULL
REPLICATIONSTATUS_STATEID: NULL
1 row in set (0.015 sec)
```

查询 TiDB 版本号

```
mysql> select tidb_version()\G
*************************** 1. row ***************************
tidb_version(): Release Version: v8.5.2
Edition: Community
Git Commit Hash: 945d07c5d5c7a1ae212f6013adfb187f2de24b23
Git Branch: HEAD
UTC Build Time: 2024-05-21 03:51:57
GoVersion: go1.21.10
Race Enabled: false
Check Table Before Drop: false
Store: tikv
1 row in set (0.001 sec)
```

4 部署

4.1 在 Kubernetes 上部署 TiDB Operator

本文介绍如何在 Kubernetes 上部署 TiDB Operator。

4.1.1 准备环境

部署 TiDB Operator 前，请确保你的环境满足以下软件要求：

- Kubernetes >= v1.30
- kubectl >= v1.30
- Helm >= v3.8

4.1.2 部署 Kubernetes 集群

TiDB Operator 运行在 Kubernetes 集群中。你可以选择以下任一方式搭建 Kubernetes 集群：

- 自托管集群：根据 Kubernetes 官方文档中任意一种方法搭建自托管的 Kubernetes 集群。
- 云服务提供商：使用 Kubernetes 认证的云服务提供商提供的 Kubernetes 集群服务。

无论选择哪种方式，请务必确保你的 Kubernetes 版本为 v1.30 或更高。如果需要快速搭建一个用于测试的简单集群，可以参考快速上手教程。

4.1.3 部署 TiDB Operator CRD

执行以下命令，安装 TiDB Operator 所需的 Custom Resource Definition (CRD)：

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.crds.yaml --server-side
```

4.1.4 部署 TiDB Operator

你可以通过以下两种方式部署 TiDB Operator：

- 使用 kubectl apply 快速部署
- 使用 Helm 部署

4.1.4.1 方式一：使用 kubectl apply 快速部署

TiDB Operator 安装所需的所有资源（包括 RBAC 和 Deployment 等，CRD 除外）都已打包在 tidb-operator.yaml 文件中。你可以使用以下命令一键部署，无需额外指定参数：

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.yaml --server-side
```

TiDB Operator 将被部署到 tidb-admin namespace 下。你可以运行以下命令验证安装是否成功：

```
kubectl get pods -n tidb-admin
```

预期输出如下：

```
NAME                            READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m
```

4.1.4.2 方式二：使用 Helm 部署

使用 Helm 部署除 CRD 外的所有资源：

```
helm install tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator --
    ↪ version v2.0.0-beta.0 --namespace tidb-admin --create-namespace
```

TiDB Operator 将被部署到 tidb-admin namespace 下。你可以运行以下命令验证安装是否成功：

```
kubectl get pods -n tidb-admin
```

预期输出如下：

```
NAME                              READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m
```

4.1.4.2.1 自定义安装

如需自定义部署参数，请先导出默认的 values.yaml 文件：

```
helm show values oci://ghcr.io/pingcap/charts/tidb-operator --version v2
    ↪ .0.0-beta.0 > values.yaml
```

根据需要修改 values.yaml，然后执行以下命令安装：

```
helm install tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator --
    ↪ version v2.0.0-beta.0 -f values.yaml
```

4.2 在 Kubernetes 上部署 TiDB 集群

本文介绍如何在 Kubernetes 环境中部署 TiDB 集群。

4.2.1 前提条件

- TiDB Operator 部署完成。

4.2.2 配置 TiDB 集群

TiDB 集群包含以下组件，每个组件由对应的 Custom Resource Definition (CRD) 进行管理：

组件名称	CRD 名称
PD	PDGroup
TiKV	TiKVGroup
TiDB	TiDBGroup
TiProxy（可选）	TiProxyGroup
TiFlash（可选）	TiFlashGroup
TiCDC（可选）	TiCDCGroup

在下面的步骤中，你将通过 Cluster CRD 定义一个 TiDB 集群，然后在各组件的 CRD 配置中，通过指定以下 cluster.name 字段将其关联到该集群。

```
spec:
  cluster:
    name: <cluster>
```

在部署 TiDB 集群之前，你需要为每个组件准备对应的 YAML 配置文件，以下是部分配置示例：

- PD 组件：pd.yaml
- TiKV 组件：tikv.yaml
- TiDB 组件：tidb.yaml
- TiFlash 组件：tiflash.yaml
- TiProxy 组件：tiproxy.yaml
- TiCDC 组件：ticdc.yaml

4.2.2.1 设置组件版本

通过 version 字段指定组件版本：

```
spec:
  template:
    spec:
      version: v8.5.2
```

如需使用非官方镜像，可通过 image 字段指定：

```
spec:
  template:
    spec:
      version: v8.5.2
      image: gcr.io/xxx/tidb
```

如果要使用的版本不符合语义化版本 (Semantic Version) 格式，也可通过 image 字段指定：

```
spec:
  template:
    spec:
      version: v8.5.2
      image: gcr.io/xxx/tidb:dev
```

> **注意：**
>
> TiDB Operator 会根据 version 字段判断组件之间的升级依赖关系。为避免升级失败，请确保指定的镜像版本准确无误。

4.2.2.2 配置资源

通过 spec.resources 字段配置组件所需的资源：

```
spec:
  resources:
    cpu: "4"
    memory: 8Gi
```

注意：

- 默认情况下，配置的资源会同时应用于 Requests 和 Limits，即 Requests 与 Limits 使用相同的资源配置。
- 如需分别设置 Requests 和 Limits，请使用Overlay 进行配置。

4.2.2.3 配置组件参数

通过 spec.config 字段设置组件的 config.toml 参数：

```
spec:
  config: |
    [log]
    level = warn
```

注意：

暂不支持校验 config.toml 配置的合法性，请确保配置内容正确。

4.2.2.4 配置存储卷

通过 spec.volumes 为组件配置挂载的存储卷 (volume)：

```
spec:
  template:
    spec:
      volumes:
      - name: test
        mounts:
        - mountPath: "/test"
        storage: 100Gi
```

部分组件支持使用 type 字段指定特定用途的 volume。此时，config.toml 中的相关配置也会自动更新，例如：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
...
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        # data is for TiKV's data dir
        - type: data
        storage: 100Gi
```

此外，volume 支持指定 StorageClass 和 VolumeAttributeClass。详情参考存储卷配置。

4.2.2.5 配置调度策略

通过 spec.schedulePolicies 字段将组件均匀分布到不同节点：

```
spec:
  schedulePolicies:
  - type: EvenlySpread
    evenlySpread:
      topologies:
      - topology:
          topology.kubernetes.io/zone: us-west-2a
      - topology:
          topology.kubernetes.io/zone: us-west-2b
      - topology:
          topology.kubernetes.io/zone: us-west-2c
```

如需为拓扑设置权重，可设置 weight 字段：

```
spec:
  schedulePolicies:
  - type: EvenlySpread
    evenlySpread:
      topologies:
      - weight: 2
        topology:
          topology.kubernetes.io/zone: us-west-2a
      - topology:
          topology.kubernetes.io/zone: us-west-2b
```

你还可以使用Overlay 配置以下调度选项：

- NodeSelector
- Toleration
- Affinity
- TopologySpreadConstraints

4.2.3 部署 TiDB 集群

在准备好 TiDB 集群各组件的 YAML 配置文件后，按照以下步骤部署 TiDB 集群：

1. 创建命名空间 Namespace：

> **注意：**
> 暂不支持跨 Namespace 引用 `Cluster`。请确保所有组件部署在同一个 Kubernetes Namespace 中。

```
kubectl create namespace db
```

2. 部署 TiDB 集群：

   方法一：各个组件分别部署（以部署一个包含 PD、TiKV 和 TiDB 组件的 TiDB 集群为例）

   `Cluster` CRD 的示例配置如下：

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: basic
  namespace: db
spec: {}
```

   创建 `Cluster`：

```
kubectl apply -f cluster.yaml --server-side
```

   PD 组件的示例配置如下：

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
  namespace: db
spec:
```

```
  cluster:
    name: basic
  replicas: 3
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 20Gi
```

创建 PD 组件：

```
kubectl apply -f pd.yaml --server-side
```

TiKV 组件的示例配置如下：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 3
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

创建 TiKV 组件：

```
kubectl apply -f tikv.yaml --server-side
```

TiDB 组件的示例配置如下：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: tidb
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 2
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
```

创建 TiDB 组件：

```
kubectl apply -f tidb.yaml --server-side
```

方法二：将以上各组件的 YAML 文件保存在本地目录中，然后使用以下命令一次性部署 TiDB 集群

```
kubectl apply -f ./<directory> --server-side
```

3. 查看 TiDB 集群各组件的运行状态：

```
kubectl get cluster -n db
kubectl get group -n db
```

4.3 访问 Kubernetes 上的 TiDB 集群

本文介绍如何通过 Kubernetes Service 访问 TiDB 集群。根据不同的访问场景需求，你可以将 Service 配置为以下类型：

- `ClusterIP`：仅限 Kubernetes 集群内部访问
- `NodePort`：允许从集群外部访问（适用于测试环境）
- `LoadBalancer`：通过云平台的 LoadBalancer 特性访问（推荐用于生产环境）

4.3.1 ClusterIP

`ClusterIP` 类型的 Service 通过集群的内部 IP 暴露服务，仅支持在 Kubernetes 集群内部访问 TiDB 集群。

你可以使用以下格式之一访问 TiDB 集群：

- `basic-tidb`：仅限在同一 Namespace 内访问
- `basic-tidb.default`：支持跨 Namespace 访问
- `basic-tidb.default.svc`：支持跨 Namespace 访问

其中，`basic-tidb` 是 Service 的名称，`default` 是 Namespace 的名称，详见 Service 与 Pod 的 DNS。

每个 TiDBGroup 会自动创建一个能够访问该 TiDBGroup 所有 TiDB 的 Service。例如，TiDBGroup `tidb-0` 会创建一个内部 Service `tidb-0-tidb`。

> **注意：**
>
> 不建议直接使用默认创建的 Service 访问 TiDB。建议根据实际访问需求自行创建 Service。

以下 YAML 示例用于创建一个能够访问 Cluster `db` 下所有 TiDB 节点的 Service：

```
apiVersion: v1
kind: Service
metadata:
  name: tidb
spec:
  selector:
    pingcap.com/managed-by: tidb-operator
    pingcap.com/cluster: db
    pingcap.com/component: tidb
  ports:
    - name: mysql
      protocol: TCP
      port: 4000
      targetPort: mysql-client
```

以下 YAML 示例用于创建一个能够访问 Cluster `db` 下特定 TiDBGroup `tidb-0` 所有 TiDB 节点的 Service：

```
apiVersion: v1
kind: Service
metadata:
  name: tidb-0
spec:
  selector:
    pingcap.com/managed-by: tidb-operator
    pingcap.com/cluster: db
```

```
    pingcap.com/component: tidb
    pingcap.com/group: tidb-0
  ports:
    - name: mysql
      protocol: TCP
      port: 4000
      targetPort: mysql-client
```

4.3.2 NodePort

在没有 LoadBalancer 的环境中，可以使用 NodePort 类型的 Service 将 TiDB 集群暴露到集群外部，允许通过节点 IP 和指定端口访问 TiDB 集群。有关详细说明，请参阅 NodePort。

> **注意：**
>
> 不建议在生产环境中使用 NodePort 类型。对于云平台上的生产环境，推荐使用 LoadBalancer 类型。

配置示例如下：

```
apiVersion: v1
kind: Service
metadata:
  name: tidb-0
spec:
  type: NodePort
  selector:
    pingcap.com/managed-by: tidb-operator
    pingcap.com/cluster: db
    pingcap.com/component: tidb
    pingcap.com/group: tidb-0
  ports:
    - name: mysql
      protocol: TCP
      port: 4000
      targetPort: mysql-client
```

4.3.3 LoadBalancer

在支持 LoadBalancer 的云平台（如 Google Cloud 或 AWS）上，建议使用云平台提供的 LoadBalancer 特性暴露 TiDB 服务，以获得更好的可用性和负载均衡能力。

你可以参考以下官方文档，通过创建 LoadBalancer Service 访问 TiDB 服务：

- AWS Load Balancer Controller
- Google Cloud LoadBalancer Service
- Azure Load Balancer Service

访问 Kubernetes Service 文档，了解更多 Service 特性以及云平台 Load Balancer 支持。

5 监控与告警

5.1 TiDB 集群的监控与告警

本文介绍如何对通过 TiDB Operator 部署的 TiDB 集群进行监控及配置告警。

5.1.1 TiDB 集群的监控

TiDB 集群的监控包括两部分：监控数据采集和监控面板。你可以使用 Prometheus 或 VictoriaMetrics 等开源组件采集监控数据，然后通过 Grafana 实现监控面板的展示。

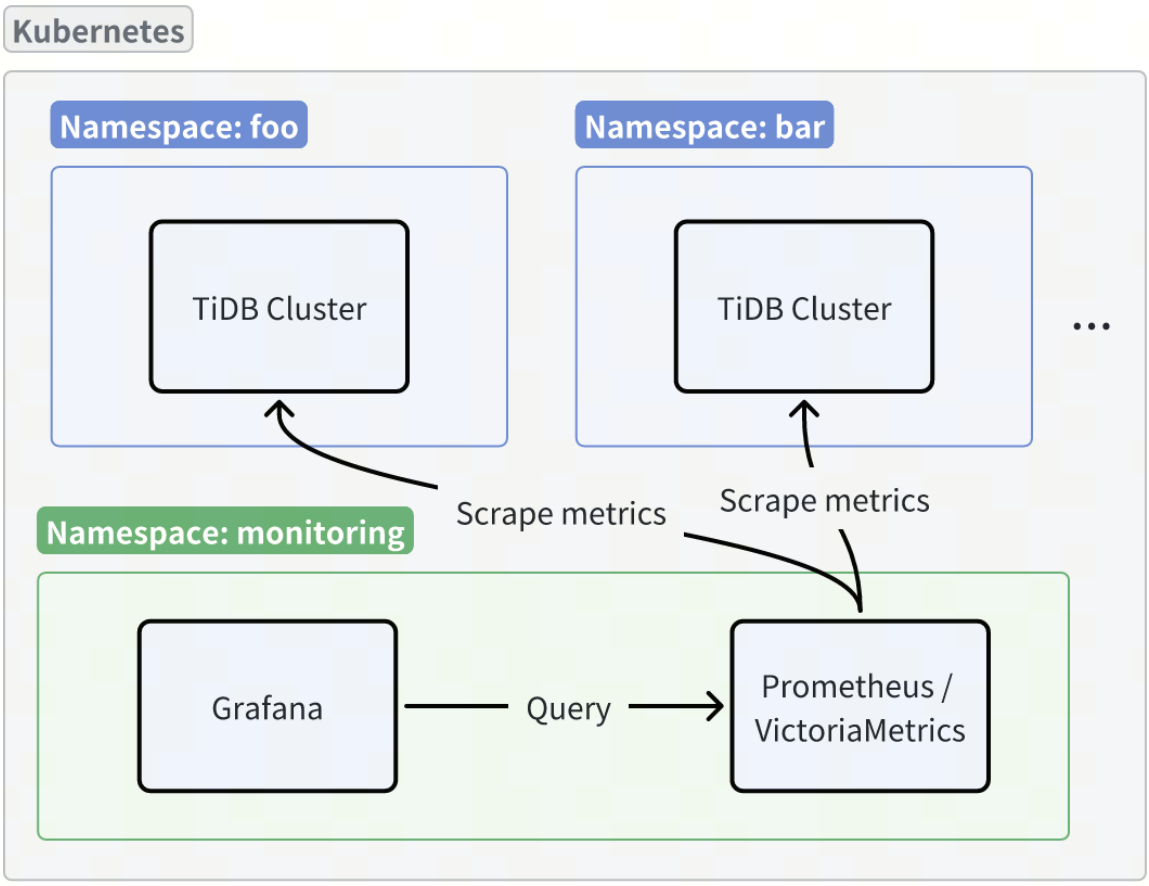

Figure 1: TiDB 集群的监控架构

5.1.1.1 监控数据采集

5.1.1.1.1 使用 Prometheus 采集监控数据

使用 Prometheus 采集监控数据的步骤如下：

1. 参考 Prometheus Operator 官方文档，在 Kubernetes 集群中部署 Prometheus Operator，本文档以 `v0.82.0` 版本为例。
2. 在每个 TiDB 集群所在的命名空间中创建一个 `PodMonitor` Custom Resource (CR)：

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: tidb-cluster-pod-monitor
  namespace: ${tidb_cluster_namespace}
```

```
  labels:
    monitor: tidb-cluster
spec:
  jobLabel: "pingcap.com/component"
  namespaceSelector:
    matchNames:
      - ${tidb_cluster_namespace}
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: tidb-operator
  podMetricsEndpoints:
    - interval: 15s
      # 若 TiDB 集群启用了 TLS，则设置为 https，否则设置为 http
      scheme: https
      honorLabels: true
      # 若 TiDB 集群启用了 TLS，则需配置 tlsConfig，否则无需配置
      tlsConfig:
        ca:
          secret:
            name: db-cluster-client-secret
            key: ca.crt
        cert:
          secret:
            name: db-cluster-client-secret
            key: tls.crt
        keySecret:
          name: db-cluster-client-secret
          key: tls.key
      metricRelabelings:
        - action: labeldrop
          regex: container
      relabelings:
        - sourceLabels: [
            ↪ __meta_kubernetes_pod_annotation_prometheus_io_scrape]
          action: keep
          regex: "true"
        - sourceLabels:
            - __meta_kubernetes_pod_name
            - __meta_kubernetes_pod_label_app_kubernetes_io_instance
            - __meta_kubernetes_pod_label_app_kubernetes_io_component
            - __meta_kubernetes_namespace
            - __meta_kubernetes_pod_annotation_prometheus_io_port
          action: replace
          regex: (.+);(.+);(.+);(.+);(.+)
          replacement: $1.$2-$3-peer.$4:$5
```

```
      targetLabel: __address__
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_annotation_prometheus_io_path]
      targetLabel: __metrics_path__
    - sourceLabels: [__meta_kubernetes_namespace]
      targetLabel: kubernetes_namespace
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_instance]
      targetLabel: cluster
    - sourceLabels: [__meta_kubernetes_pod_name]
      targetLabel: instance
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_component]
      targetLabel: component
    - sourceLabels:
        - __meta_kubernetes_namespace
        - __meta_kubernetes_pod_label_app_kubernetes_io_instance
      separator: '-'
      targetLabel: tidb_cluster
```

3. 参考 Prometheus Operator 官方文档，创建一个 Prometheus CR 用来采集监控指标，确保为 ServiceAccount 分配必要权限：

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
  namespace: monitoring
spec:
  serviceAccountName: prometheus
  externalLabels:
    k8s_cluster: ${your_k8s_cluster_name}
  podMonitorSelector:
    matchLabels:
      monitor: tidb-cluster
  # podMonitorNamespaceSelector 设置为空，表示采集所有命名空间中的
    ↪ PodMonitor
  podMonitorNamespaceSelector: {}
```

4. 运行以下 kubectl port-forward 命令，通过端口转发访问 Prometheus：

```
kubectl port-forward -n monitoring prometheus-prometheus-0 9090:9090
  ↪ &>/tmp/portforward-prometheus.log &
```

然后在浏览器中访问 http://localhost:9090/targets 查看监控数据采集状态。

5.1.1.1.2 使用 VictoriaMetrics 采集监控数据

使用 VictoriaMetrics 部署监控数据采集的步骤如下：

1. 参考 VictoriaMetrics 官方文档，在 Kubernetes 集群中部署 VictoriaMetrics Operator，本文档以 `v0.58.1` 版本为例。

2. 创建一个 `VMSingle` Custom Resource (CR) 用来存储监控指标：

```
apiVersion: victoriametrics.com/v1beta1
kind: VMSingle
metadata:
  name: demo
  namespace: monitoring
```

3. 创建一个 `VMAgent` CR 用来采集监控指标：

```
apiVersion: victoriametrics.com/v1beta1
kind: VMAgent
metadata:
  name: demo
  namespace: monitoring
spec:
  # 配置远程写入，将采集到的监控指标写入 VMSingle
  remoteWrite:
    - url: "http://vmsingle-demo.monitoring.svc:8429/api/v1/write"
  externalLabels:
    k8s_cluster: ${your_k8s_cluster_name}
  selectAllByDefault: true
```

4. 在每个 TiDB 集群所在的命名空间中创建一个 `VMPodScrape` CR，用来发现 TiDB 集群的 Pod，并为 `VMAgent` 生成相应的 scrape 配置：

```
apiVersion: victoriametrics.com/v1beta1
kind: VMPodScrape
metadata:
  name: tidb-cluster-pod-scrape
  namespace: ${tidb_cluster_namespace}
spec:
  jobLabel: "pingcap.com/component"
  namespaceSelector:
    matchNames:
      - ${tidb_cluster_namespace}
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: tidb-operator
  podMetricsEndpoints:
```

```
- interval: 15s
  # 若 TiDB 集群启用了 TLS，则设置为 https，否则设置为 http
  scheme: https
  honorLabels: true
  # 若 TiDB 集群启用了 TLS，则需配置 TLS 认证，否则无需配置
  tlsConfig:
    ca:
      secret:
        name: db-cluster-client-secret
        key: ca.crt
    cert:
      secret:
        name: db-cluster-client-secret
        key: tls.crt
    keySecret:
      name: db-cluster-client-secret
      key: tls.key
  metricRelabelConfigs:
    - action: labeldrop
      regex: container
  relabelConfigs:
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_annotation_prometheus_io_scrape]
      action: keep
      regex: "true"
    - sourceLabels:
        - __meta_kubernetes_pod_name
        - __meta_kubernetes_pod_label_app_kubernetes_io_instance
        - __meta_kubernetes_pod_label_app_kubernetes_io_component
        - __meta_kubernetes_namespace
        - __meta_kubernetes_pod_annotation_prometheus_io_port
      action: replace
      regex: (.+);(.+);(.+);(.+);(.+)
      replacement: $1.$2-$3-peer.$4:$5
      targetLabel: __address__
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_annotation_prometheus_io_path]
      targetLabel: __metrics_path__
    - sourceLabels: [__meta_kubernetes_namespace]
      targetLabel: kubernetes_namespace
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_instance]
      targetLabel: cluster
    - sourceLabels: [__meta_kubernetes_pod_name]
      targetLabel: instance
```

```
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_component]
      targetLabel: component
    - sourceLabels:
        - __meta_kubernetes_namespace
        - __meta_kubernetes_pod_label_app_kubernetes_io_instance
      separator: '-'
      targetLabel: tidb_cluster
```

5. 运行以下 `kubectl port-forward` 命令，通过端口转发访问 VMAgent：

```
kubectl port-forward -n monitoring svc/vmagent-demo 8429:8429 &>/tmp/
    ↪ portforward-vmagent.log &
```

然后在浏览器中访问 http://localhost:8429/targets 查看监控数据采集状态。

5.1.1.2 监控面板

配置监控面板的步骤如下：

1. 参考 Grafana 官方文档，在 Kubernetes 集群中部署 Grafana，本文档以 `12.0.0-security-01` 版本为例。

2. 运行以下 `kubectl port-forward` 命令，通过端口转发访问 Grafana 监控面板：

```
kubectl port-forward -n ${namespace} ${grafana_pod_name} 3000:3000 &>/
    ↪ tmp/portforward-grafana.log &
```

3. 在浏览器中访问 http://localhost:3000，默认用户名和密码都为 `admin`。如果是通过 Helm 安装，可以使用以下命令查看 `admin` 密码：

```
kubectl get secret --namespace ${namespace} ${grafana_secret_name} -o
    ↪ jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

4. 在 Grafana 中添加 Prometheus 类型的数据源，并配置 Prometheus Server URL：

   - 如果使用 Prometheus 采集监控指标，设置 URL 为 `http://prometheus-operated.monitoring.svc:9090`。
   - 如果使用 VictoriaMetrics 采集监控指标，设置 URL 为 `http://vmsingle-demo.monitoring.svc:8429`。

5. 可以使用 `get-grafana-dashboards.sh` 脚本下载各组件的监控面板，然后手动导入到 Grafana 中。

5.1.2 告警配置

你可以通过 AlertManager 管理与发送告警信息，具体的部署和配置步骤请参考 Alertmanager 官方文档。

5.1.3 使用 Grafana 查看多集群监控

要使用 Grafana 查看多个集群的监控，请在每个 Grafana Dashboard 中进行以下操作：

1. 在 Grafana Dashboard 中，点击 **Dashboard settings** 选项，打开 **Settings** 页面。
2. 在 **Settings** 页面中，选择 **Variables** 中的 **tidb_cluster** 变量，将 **tidb_cluster** 变量的 **Hide** 属性设置为空选项。
3. 返回当前 Grafana Dashboard，即可看到集群选择下拉框。下拉框中的集群名称格式为 `${namespace}-${tidb_cluster_name}`。
4. 点击 **Save dashboard** 保存对该 Dashboard 的修改。

5.2 Kubernetes 可观测性：监控、告警与日志收集

本文介绍如何在 Kubernetes 集群中进行监控、告警与日志收集，帮助你全面掌握集群及其组件的运行状态。

5.2.1 监控

5.2.1.1 TiDB 组件监控

随集群部署的 TiDB 监控只关注 TiDB 本身各组件的运行情况，并不包括对容器资源、宿主机、Kubernetes 组件和 TiDB Operator 等的监控。对于这些组件或资源的监控，需要在整个 Kubernetes 集群维度部署监控系统来实现。

5.2.1.2 宿主机监控

对宿主机及其资源的监控与传统的服务器物理资源监控相同。

如果在你的现有基础设施中已经有针对物理服务器的监控系统，只需要通过常规方法将 Kubernetes 所在的宿主机添加到现有监控系统中即可；如果没有可用的监控系统，或者希望部署一套独立的监控系统用于监控 Kubernetes 所在的宿主机，可以使用你熟悉的任意监控系统。

新部署的监控系统可以运行于独立的服务器、直接运行于 Kubernetes 所在的宿主机，也可以运行于 Kubernetes 集群内。虽然不同部署方式在安装配置与资源利用上存在少许差异，但是在使用上并没有重大区别。

常见的可用于监控服务器资源的开源监控系统有：

- Prometheus 和 node_exporter
- VictoriaMetrics

- CollectD
- Nagios
- Zabbix

一些云服务商或专门的性能监控服务提供商也有各自的免费或收费的监控解决方案可以选择。

推荐通过 Prometheus Operator 在 Kubernetes 集群内部署基于 Node Exporter 和 Prometheus 的宿主机监控系统，这一方案同时可以兼容并用于 Kubernetes 自身组件的监控。

5.2.1.3 Kubernetes 组件监控

对 Kubernetes 组件的监控可以参考官方文档提供的方案，也可以使用其他兼容 Kubernetes 的监控系统来进行。

一些云服务商可能提供了自己的 Kubernetes 组件监控方案，一些专门的性能监控服务商也有各自的 Kubernetes 集成方案可以选择。

由于 TiDB Operator 实际上是运行于 Kubernetes 中的容器，选择任一可以覆盖对 Kubernetes 容器状态及资源进行监控的监控系统即可覆盖对 TiDB Operator 的监控，无需再额外部署监控组件。

推荐通过 Prometheus Operator 部署基于 Node Exporter 和 Prometheus 的宿主机监控系统，这一方案同时可以兼容并用于对宿主机资源的监控。

5.2.2 告警

如果使用 Prometheus Operator 部署针对 Kubernetes 宿主机和服务的监控，会默认配置一些告警规则，并且会部署一个 AlertManager 服务，具体的设置方法请参阅 kube-prometheus 的说明。

如果使用其他的工具或服务对 Kubernetes 宿主机和服务进行监控，请查阅该工具或服务提供商的对应资料。

5.2.3 日志收集

5.2.3.1 TiDB 与 Kubernetes 组件运行日志

通过 TiDB Operator 部署的 TiDB 各组件默认将运行日志输出到容器的 `stdout` 和 `stderr`。在 Kubernetes 环境中，这些日志存储在宿主机的 `/var/log/containers` 目录下，文件名包含 Pod 和容器名称等信息。因此，你可以直接在宿主机上收集容器中应用的日志。

如果你现有的基础设施已具备日志收集系统，只需要通过常规方法将 Kubernetes 所在的宿主机上的 `/var/log/containers/*.log` 文件加入采集范围即可。如果没有可用的日志收集系统，或者希望部署一套独立的系统用于收集相关日志，可以使用你熟悉的任何日志收集系统或方案。

常见的可用于收集 Kubernetes 日志的开源工具有：

- Vector
- Fluentd
- Fluent Bit
- Filebeat
- Logstash

收集到的日志通常可以汇总存储在某一特定的服务器上，或存放到 Elasticsearch 等专用的存储和分析系统中。

此外，一些云服务商或性能监控服务提供商也提供了免费或付费的日志收集方案。

5.2.3.2 系统日志

系统日志可以通过常规方法在 Kubernetes 宿主机上收集。如果你现有的基础设施已具备日志收集系统，只需要通过常规方法将相关服务器和日志文件添加到收集范围即可。如果没有可用的日志收集系统，或者希望部署一套独立的系统用于收集相关日志，可以使用你熟悉的任何日志收集系统或方案。

TiDB 与 Kubernetes 组件运行日志章节提到的几种常见日志收集工具均支持对系统日志的收集。此外，一些云服务商或性能监控服务提供商也提供了免费或付费的日志收集方案。

6 集群配置

6.1 组件配置

本文档介绍如何配置 TiDB、TiKV、PD、TiProxy、TiFlash、TiCDC 等组件的配置参数。

TiDB Operator 默认通过滚动重启相关组件，使配置生效。

6.1.1 配置 TiDB 配置参数

你可以通过 TiDBGroup CR 的 `spec.template.spec.config` 来配置 TiDB 配置参数。

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: tidb
spec:
  template:
    spec:
      config: |
        split-table = true
        oom-action = "log"
```

获取所有可以配置的 TiDB 配置参数，请参考 TiDB 配置文档。

6.1.2 配置 TiKV 配置参数

你可以通过 TiKVGroup CR 的 spec.template.spec.config 来配置 TiKV 配置参数。

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      config: |
        [storage]
          [storage.block-cache]
            capacity = "16GB"
        [log.file]
          max-days = 30
          max-backups = 30
```

获取所有可以配置的 TiKV 配置参数，请参考 TiKV 配置文档。

> **注意：**
>
> TiKV 的 RocksDB 日志默认存储在 /var/lib/tikv 数据目录，建议配置 max-days 和 max-backups 来自动清理日志文件。

6.1.3 配置 PD 配置参数

你可以通过 PDGroup CR 的 spec.template.spec.config 来配置 PD 配置参数。

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      config: |
        lease = 3
        enable-prevote = true
```

获取所有可以配置的 PD 配置参数，请参考 PD 配置文档。

注意：

PD 部分配置项在首次启动成功后会持久化到 etcd 中且后续将以 etcd 中的配置为准。因此 PD 在首次启动后，这些配置项将无法再通过配置参数来进行修改，而需要使用 SQL、pd-ctl 或 PD server API 来动态进行修改。目前，在线修改 PD 配置文档中所列的配置项中，除 `log.level` 外，其他配置项在 PD 首次启动之后均不再支持通过配置参数进行修改。

6.1.3.1 配置 PD 微服务

注意：

- PD 从 v8.0.0 版本开始支持微服务模式。
- 目前只支持在创建时开启 PD 微服务模式，后续无法再修改该字段。

你可以通过设置 PDGroup CR 的 `spec.template.spec.mode` 为 `"ms"` 来开启 PD 微服务模式：

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      mode: "ms"
```

目前 PD 支持 `tso` 和 `scheduling` 这两个微服务，你可以通过 TSOGroup 和 SchedulerGroup CR 的 `spec.template.spec.config` 来配置 PD 微服务参数。

```
apiVersion: core.pingcap.com/v1alpha1
kind: TSOGroup
metadata:
  name: tso
spec:
  template:
    spec:
      config: |
        [log.file]
```

```
        filename = "/pdms/log/tso.log"
---
apiVersion: core.pingcap.com/v1alpha1
kind: SchedulerGroup
metadata:
  name: scheduling
spec:
  template:
    spec:
      config: |
        [log.file]
          filename = "/pdms/log/scheduling.log"
```

关于 PD 微服务、tso 组件和 scheduling 组件的完整配置参数，请参考以下文档：

- PD 配置文件描述
- TSO 配置文件描述
- Scheduling 配置文件描述

> 注意：
>
> - 如果在部署 TiDB 集群时就启用了 PD 微服务模式，PD 微服务的部分配置项会持久化到 etcd 中且后续将以 etcd 中的配置为准。
> - 因此，PD 微服务在首次启动后，这些配置项将无法再通过配置参数来进行修改，而需要使用 SQL、pd-ctl 或 PD server API 来动态进行修改。目前，在线修改 PD 配置文档中所列的配置项中，除 log.level 外，其他配置项在 PD 微服务首次启动之后均不再支持通过配置参数进行修改。

6.1.4 配置 TiProxy 配置参数

你可以通过 TiProxyGroup CR 的 spec.template.spec.config 来配置 TiProxy 配置参数。

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiProxyGroup
metadata:
  name: tiproxy
spec:
  template:
    spec:
```

```
    config: |
      [log]
        level = "info"
```

获取所有可以配置的 TiProxy 配置参数，请参考 TiProxy 配置文档。

6.1.5 配置 TiFlash 配置参数

你可以通过 TiFlashGroup CR 的 `spec.template.spec.config` 来配置 TiFlash 配置参数。

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiFlashGroup
metadata:
  name: tiflash
spec:
  template:
    spec:
      config: |
        [flash]
          [flash.flash_cluster]
            log = "/data0/logs/flash_cluster_manager.log"
        [logger]
          count = 10
          level = "information"
          errorlog = "/data0/logs/error.log"
          log = "/data0/logs/server.log"
```

获取所有可以配置的 TiFlash 配置参数，请参考 TiFlash 配置文档。

6.1.6 配置 TiCDC 启动参数

你可以通过 TiCDCGroup CR 的 `spec.template.spec.config` 来配置 TiCDC 启动参数。

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiCDCGroup
metadata:
  name: ticdc
spec:
  template:
    spec:
      config: |
        gc-ttl = 86400
        log-level = "info"
```

获取所有可以配置的 TiCDC 启动参数，请参考 TiCDC 启动参数文档。

6.2 存储卷配置

本文档介绍如何在 TiDB Operator 中为 TiDB 集群的各组件配置存储卷，以及如何修改已创建的存储卷。

6.2.1 概述

在 TiDB Operator 中，存储卷 (Volume) 为 TiDB 集群中的各组件提供持久化存储。PD、TiKV 和 TiFlash 等组件都需要配置存储卷以保存其数据。存储卷配置的结构如下：

```
volumes:
- name: <卷名称>
  mounts:
  - type: <挂载类型>
    mountPath: <挂载路径>
    subPath: <子路径>
  storage: <存储大小>
  storageClassName: <存储类名称>
  volumeAttributesClassName: <卷属性类名称>
```

参数说明：

- `name`：卷的名称，在同一组件内必须唯一。
- `mounts`：定义卷的挂载信息，包括：
  - `type`：指定挂载类型，不同组件支持的类型不同。
  - `mountPath`：（可选）指定挂载路径。如果未指定，将使用默认路径。
  - `subPath`：（可选）指定卷内子路径。
- `storage`：存储容量，例如 `100Gi`。
- `storageClassName`：（可选）指定 Kubernetes 存储类名称。
- `volumeAttributesClassName`：（可选）指定卷属性类，用于修改卷的属性。此功能仅 Kubernetes 1.29 及以上版本支持。

6.2.2 组件特定的存储卷配置

6.2.2.1 PD 存储卷配置

PD 组件支持的挂载类型：

- `data`：PD 数据目录，默认路径为 `/var/lib/pd`

示例配置：

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 20Gi
```

6.2.2.2 TiKV 存储卷配置

TiKV 组件支持的挂载类型：

- data：TiKV 数据目录，默认路径为 /var/lib/tikv

示例配置：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

6.2.2.3 TiDB 存储卷配置

TiDB 组件支持的挂载类型：

- data：TiDB 数据目录，默认路径为 /var/lib/tidb
- slowlog：TiDB 慢日志目录，默认路径为 /var/log/tidb

示例配置：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: tidb
spec:
  template:
    spec:
      volumes:
      - name: slowlog
        mounts:
          - type: slowlog
        storage: 10Gi
```

6.2.2.4 TiFlash 存储卷配置

TiFlash 组件支持的挂载类型：

- `data`：TiFlash 数据目录，默认路径为 `/var/lib/tiflash`

示例配置：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiFlashGroup
metadata:
  name: tiflash
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

6.2.3 修改存储卷

6.2.3.1 修改存储大小

通过修改组件 Group CR 中 `volumes.storage` 字段，TiDB Operator 会自动更新对应的 PVC 以调整存储大小。

注意：

- 只有当所使用的 StorageClass 的 `allowVolumeExpansion` 设置为 `true` 时，才能修改存储大小。
- 只支持扩容，不支持缩容。

6.2.3.2 修改存储卷属性

对于底层使用云厂商存储的 TiDB 集群，TiDB Operator 支持以下两种方式修改存储卷的属性（例如 IOPS 和吞吐量）：

- （推荐）Kubernetes 原生方式
- 云厂商 API 方式

6.2.3.2.1 方式一：Kubernetes 原生方式

Kubernetes 原生方式是指通过 Volume Attributes Classes 功能修改卷的属性。

前提条件：

- Kubernetes 1.29 及以上版本。
- 已开启 Kubernetes 的 Volume Attributes Classes 功能。

使用 Kubernetes 原生方式修改卷属性的操作步骤如下：

1. 在 TiDB 集群中启用 `VolumeAttributesClass` 功能开关：

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: basic
spec:
  featureGates:
    - name: VolumeAttributesClass
```

2. 创建 `VolumeAttributesClass` 资源：

```
apiVersion: storage.k8s.io/v1beta1
kind: VolumeAttributesClass
metadata:
  name: silver
driverName: pd.csi.storage.gke.io
parameters:
  provisioned-iops: "3000"
  provisioned-throughput: "50"
```

3. 通过修改组件 Group CR 中的 `volumes.volumeAttributesClassName` 字段来使用步骤 2 中创建的 `VolumeAttributesClass`，从而修改卷的属性：

```
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
        volumeAttributesClassName: silver
```

6.2.3.2.2 方式二：云厂商 API

> **注意：**
>
> 使用云厂商 API 方式时，你需要为 TiDB Operator 配置相应的云厂商权限。

未启用 `VolumeAttributesClass` 功能时，TiDB Operator 会调用云厂商的 API 直接修改存储卷属性。目前支持对以下云厂商存储卷的修改：

- **AWS EBS**：使用 AWS EC2 API 修改 EBS 卷的大小、IOPS 和吞吐量。
- **Azure Disk**：使用 Azure API 修改 Managed Disk 的大小、IOPS 和吞吐量。

以 AWS EBS 为例，假设当前存储卷的配置为：

```
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
        storageClassName: gp3-2000-100
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3-2000-100
parameters:
```

```
  csi.storage.k8s.io/fstype: ext4
  encrypted: "true"
  iops: "2000"
  throughput: "100"
  type: gp3
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

你可以创建一个更高 IOPS 和吞吐量的 StorageClass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3-4000-400
parameters:
  csi.storage.k8s.io/fstype: ext4
  encrypted: "true"
  iops: "4000"
  throughput: "400"
  type: gp3
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

然后，将组件 Group CR 中的 `.spec.template.spec.volumes.storageClassName` 字段修改为 gp3-4000-400，TiDB Operator 会自动调用云厂商 API 执行修改。

由于 PVC 的 StorageClass 不能直接修改，TiDB Operator 会在 PVC 上添加以下注解来跟踪修改状态：

- `spec.tidb.pingcap.com/revision`：期望的修改版本号
- `spec.tidb.pingcap.com/storage-class`：期望的存储类
- `spec.tidb.pingcap.com/storage-size`：期望的存储大小
- `status.tidb.pingcap.com/revision`：当前的修改版本号
- `status.tidb.pingcap.com/storage-class`：当前的存储类
- `status.tidb.pingcap.com/storage-size`：当前的存储大小

6.2.4 常见问题

6.2.4.1 如何为不同的 TiKV 实例配置不同的存储大小？

同一个 TiKVGroup 中的所有实例使用相同的存储配置。如果你需要为不同的 TiKV 实例配置不同的存储，可以创建多个 TiKVGroup。

6.2.4.2 为什么存储卷修改没有立即生效？

存储卷修改未立即生效可能有以下原因：

- 部分云厂商对卷修改存在冷却期限制，例如 AWS EBS 有 6 小时冷却期。
- 文件系统扩容可能需要一些时间才能完成。

6.3 自定义 Kubernetes 原生资源的配置

本文介绍如何使用 TiDB Operator 提供的 Overlay 功能，自定义 Kubernetes 原生资源配置。

Overlay 功能是 TiDB Operator 提供的一种配置机制，你无需修改 TiDB Operator 源代码，就可以自定义 Kubernetes 集群中原生资源的配置（例如 Pod 和 PersistentVolumeClaim (PVC) 等），从而满足特定的部署需求。

6.3.1 支持的资源类型

目前，TiDB Operator 支持通过 Overlay 功能自定义以下资源类型：

- **Pod**：修改 Pod 的元数据（如标签、注解）和规约（`spec` 字段）
- **PersistentVolumeClaim (PVC)**：修改 PVC 的元数据（如标签、注解）

> **注意：**
>
> 暂不支持修改 PVC 的规约 (`spec`)。

6.3.2 使用方法

6.3.2.1 自定义 Pod 配置 (Pod Overlay)

Pod Overlay 可以修改 Pod 的元数据（如标签、注解）和规约 (`spec`) 配置。你可以在 Component Group（例如 `PDGroup`、`TiDBGroup`、`TiKVGroup`、`TiFlashGroup`、`TiProxyGroup`、`TiCDCGroup` 等）的 Custom Resource (CR) 中使用 `spec.template.spec.overlay.pod` 字段进行配置。

以下示例展示如何为 PD 容器添加一个名为 `CUSTOM_ENV_VAR` 的环境变量：

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
```

```
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: pd
                env:
                  - name: "CUSTOM_ENV_VAR"
                    value: "custom_value"
```

6.3.2.2 自定义 PVC 配置 (PVC Overlay)

PVC Overlay 可以修改 PVC 的元数据 (如标签、注解)。你可以在 Component Group (例如 PDGroup、TiDBGroup、TiKVGroup、TiFlashGroup、TiProxyGroup、TiCDCGroup 等) 的 Custom Resource (CR) 中使用 spec.template.spec.overlay.volumeClaims 字段进行配置。

以下示例展示如何为 TiKV 的 PVC 添加一个名为 custom-label 的自定义标签：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
          - type: data
        storage: 100Gi
      overlay:
        volumeClaims:
          - name: data
            volumeClaim:
              metadata:
                labels:
                  custom-label: "value"
```

6.3.3 注意事项

在使用 Overlay 功能时，请注意以下事项：

- Overlay 功能会将你在 `spec.template.spec.overlay` 字段中定义的配置与 TiDB Operator 自动生成的默认配置进行合并。如果存在冲突，以 Overlay 中定义的配置为准。
- 对于 `nodeSelector`、`labels` 等 map 类型的字段，Overlay 功能会在保留原有配置的基础上追加你定义的键值对，而非完全替换。
- 在使用 Overlay 功能修改配置前，请务必充分了解这些修改可能带来的影响，特别是涉及安全上下文 (`securityContext`) 和资源限制等关键配置时。
- 支持通过 Overlay 进行修改的字段取决于 TiDB Operator 依赖的 Kubernetes API 版本。

6.3.4 示例场景

本节提供一些常见的 Overlay 使用示例，帮助你快速理解并应用该功能来自定义 Kubernetes 资源配置。

6.3.4.1 配置 Pod 的资源限制和亲和性

以下示例展示如何使用 Overlay 配置 TiDB Pod 的资源请求限制及亲和性 (affinity)：

```
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tidb
                resources:
                  limits:
                    cpu: "4"
                    memory: "8Gi"
                  requests:
                    cpu: "2"
                    memory: "4Gi"
            affinity:
              nodeAffinity:
                requiredDuringSchedulingIgnoredDuringExecution:
                  nodeSelectorTerms:
                  - matchExpressions:
                    - key: dedicated
                      operator: In
                      values:
                      - tidb
```

6.3.4.2 配置 Pod 的安全上下文

以下示例演示如何通过 Overlay 配置 Pod 的 sysctls 参数：

```
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            securityContext:
              sysctls:
              - name: net.core.somaxconn
                value: "1024"
```

6.3.4.3 原地更新 Pod 和 PVC 的标签或注解

通过 Overlay，可以在不重启 Pod 的情况下，原地更新 Pod 和 PVC 的标签或注解：

```
spec:
  template:
    spec:
      overlay:
        pod:
          metadata:
            labels:
              custom-label: "value"
            annotations:
              custom-annotation: "value"
        volumeClaims:
          - name: data
            volumeClaim:
              metadata:
                labels:
                  custom-label: "value"
                annotations:
                  custom-annotation: "value"
```

6.3.4.4 注入 sidecar 容器

可以通过 Overlay 向 Pod 中添加一个 sidecar 容器，例如用于监控或日志收集：

```
spec:
  template:
    spec:
      overlay:
```

```
    pod:
      spec:
        initContainers:
          - name: logshipper
            image: alpine:latest
            restartPolicy: Always
            command: ['sh', '-c', 'tail -F /opt/logs.txt']
```

7 运维管理

7.1 安全

7.1.1 为 MySQL 客户端开启 TLS

本文主要描述了在 Kubernetes 上如何为 TiDB 集群的 MySQL 客户端开启 TLS。开启步骤为：

1. 为 TiDB Server 颁发一套 Server 端证书，为 MySQL Client 颁发一套 Client 端证书。并创建两个 Secret 对象，Secret 名字分别为：`${tidb_group_name}-tidb-server-secret` 和 `${tidb_group_name}-tidb-client-secret`，分别包含前面创建的两套证书；

> 注意：
>
> - 创建的 Secret 对象必须符合上述命名规范，否则将导致 TiDB 集群部署失败。
> - 显式指定 MySQL TLS Secret 的功能将在后续版本中支持。
> - TiDB Operator v2 与 v1 的 Secret 的默认命名方式不同：
>   - 对于 TiDB Operator v1 创建的 TiDB 集群，Secret 的默认命名为 `${cluster_name}-tidb-server-secret` 和 `${cluster_name}-tidb-client-secret`。
>   - 在 TiDB Operator v2 中，不同的 `TiDBGroup` 支持使用不同的 TLS 证书，因此 Secret 的默认命名 `${tidb_group_name}-tidb-server-secret` 和 `${tidb_group_name}-tidb-client-secret`。

2. 部署集群，设置 `TiDBGroup` 的 `.spec.template.spec.security.tls.mysql.enabled` 属性为 `true`：

> **注意：**
> 开启或变更已部署的 TiDBGroup 的 TLS 配置，将导致 TiDB Pod 滚动重启，请谨慎操作。

3. 配置 MySQL 客户端使用加密连接。

其中，颁发证书的方式有多种，本文档提供两种方式，用户也可以根据需要为 TiDB 集群颁发证书，这两种方式分别为：

- 使用 cfssl 系统颁发证书；
- （推荐）使用 cert-manager 系统颁发证书；

当需要更新已有 TLS 证书时，可参考更新和替换 TLS 证书。

7.1.1.1 第一步：为 TiDB 集群颁发两套证书

7.1.1.1.1 使用 cfssl 系统颁发证书

1. 首先下载 cfssl 软件并初始化证书颁发机构：

```
mkdir -p ~/bin
curl -s -L -o ~/bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
curl -s -L -o ~/bin/cfssljson https://pkg.cfssl.org/R1.2/
    ↪ cfssljson_linux-amd64
chmod +x ~/bin/{cfssl,cfssljson}
export PATH=$PATH:~/bin

mkdir -p cfssl
cd cfssl
cfssl print-defaults config > ca-config.json
cfssl print-defaults csr > ca-csr.json
```

2. 在 ca-config.json 配置文件中配置 CA 选项：

```
{
    "signing": {
        "default": {
            "expiry": "8760h"
        },
        "profiles": {
            "server": {
                "expiry": "8760h",
```

```
            "usages": [
                "signing",
                "key encipherment",
                "server auth"
            ]
        },
        "client": {
            "expiry": "8760h",
            "usages": [
                "signing",
                "key encipherment",
                "client auth"
            ]
        }
    }
}
```

3. 您还可以修改 ca-csr.json 证书签名请求 (CSR)：

```
{
    "CN": "TiDB Server",
    "CA": {
        "expiry": "87600h"
    },
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "US",
            "L": "CA",
            "O": "PingCAP",
            "ST": "Beijing",
            "OU": "TiDB"
        }
    ]
}
```

4. 使用定义的选项生成 CA：

```
cfssl gencert -initca ca-csr.json | cfssljson -bare ca -
```

5. 生成 Server 端证书。

首先生成默认的 server.json 文件：

```
cfssl print-defaults csr > server.json
```

然后编辑这个文件，修改 CN，hosts 属性：

```
...
    "CN": "TiDB Server",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${tidb_group_name}-tidb",
      "${tidb_group_name}-tidb.${namespace}",
      "${tidb_group_name}-tidb.${namespace}.svc",
      "*.${tidb_group_name}-tidb",
      "*.${tidb_group_name}-tidb.${namespace}",
      "*.${tidb_group_name}-tidb.${namespace}.svc",
      "*.${tidb_group_name}-tidb-peer",
      "*.${tidb_group_name}-tidb-peer.${namespace}",
      "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
    ],
...
```

其中 ${tidb_group_name} 为 TiDBGroup 的名字，${namespace} 为 TiDB 集群部署的命名空间，用户也可以添加自定义 hosts。

最后生成 Server 端证书：

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -
    ↪ profile=server server.json | cfssljson -bare server
```

6. 生成 Client 端证书。

首先生成默认的 client.json 文件：

```
cfssl print-defaults csr > client.json
```

然后编辑这个文件，修改 CN，hosts 属性，hosts 可以留空：

```
...
    "CN": "TiDB Client",
    "hosts": [],
...
```

最后生成 Client 端证书：

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -
    ↪ profile=client client.json | cfssljson -bare client
```

7. 创建 Kubernetes Secret 对象。

   到这里假设你已经按照上述文档把两套证书都创建好了。通过下面的命令为 TiDB 集群创建 Secret 对象：

```
kubectl create secret generic ${tidb_group_name}-tidb-server-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=server.pem --from-file
    ↪ =tls.key=server-key.pem --from-file=ca.crt=ca.pem
kubectl create secret generic ${tidb_group_name}-tidb-client-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=client.pem --from-file
    ↪ =tls.key=client-key.pem --from-file=ca.crt=ca.pem
```

   这样就给 Server/Client 端证书分别创建了：

   - 一个 Secret 供 TiDB Server 启动时加载使用;
   - 另一个 Secret 供 MySQL 客户端连接 TiDB 集群时候使用。

用户可以生成多套 Client 端证书，并且至少要生成一套 Client 证书供 TiDB Operator 内部组件访问 TiDB Server。

7.1.1.1.2 使用 cert-manager 颁发证书

1. 安装 cert-manager。

   请参考官网安装：cert-manager installation on Kubernetes。

2. 创建一个 Issuer 用于给 TiDB 集群颁发证书。

   为了配置 `cert-manager` 颁发证书，必须先创建 Issuer 资源。

   首先创建一个目录保存 `cert-manager` 创建证书所需文件：

```
mkdir -p cert-manager
cd cert-manager
```

   然后创建一个 `tidb-server-issuer.yaml` 文件，输入以下内容：

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ${cluster_name}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${cluster_name}-ca
```

```
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-ca-secret
  commonName: "TiDB CA"
  isCA: true
  duration: 87600h # 10yrs
  renewBefore: 720h # 30d
  issuerRef:
    name: ${cluster_name}-selfsigned-ca-issuer
    kind: Issuer
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ${cluster_name}-cert-issuer
  namespace: ${namespace}
spec:
  ca:
    secretName: ${cluster_name}-ca-secret
```

上面的文件创建三个对象：

- 一个 SelfSigned 类型的 Issuer 对象（用于生成 CA 类型 Issuer 所需要的 CA 证书）；
- 一个 Certificate 对象，isCa 属性设置为 true；
- 一个可以用于颁发 TiDB Server TLS 证书的 Issuer。

最后执行下面的命令进行创建：

```
kubectl apply -f tidb-server-issuer.yaml
```

3. 创建 Server 端证书。

   在 cert-manager 中，Certificate 资源表示证书接口，该证书将由上面创建的 Issuer 颁发并保持更新。

   首先来创建 Server 端证书，创建一个 tidb-server-cert.yaml 文件，并输入以下内容：

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tidb_group_name}-tidb-server-secret
  namespace: ${namespace}
spec:
  secretName: ${tidb_group_name}-tidb-server-secret
  duration: 8760h # 365d
```

```
renewBefore: 360h # 15d
subject:
  organizations:
  - PingCAP
commonName: "TiDB"
usages:
  - server auth
dnsNames:
  - "${tidb_group_name}-tidb"
  - "${tidb_group_name}-tidb.${namespace}"
  - "${tidb_group_name}-tidb.${namespace}.svc"
  - "*.${tidb_group_name}-tidb"
  - "*.${tidb_group_name}-tidb.${namespace}"
  - "*.${tidb_group_name}-tidb.${namespace}.svc"
  - "*.${tidb_group_name}-tidb-peer"
  - "*.${tidb_group_name}-tidb-peer.${namespace}"
  - "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
ipAddresses:
  - 127.0.0.1
  - ::1
issuerRef:
  name: ${cluster_name}-cert-issuer
  kind: Issuer
  group: cert-manager.io
```

其中 ${cluster_name} 为集群的名字，${tidb_group_name} 为 TiDBGroup 的名字：

- spec.secretName 请设置为 ${tidb_group_name}-tidb-server-secret；
- usages 请添加上 server auth；
- dnsNames 需要填写这 6 个 DNS，根据需要可以填写其他 DNS：
- ${tidb_group_name}-tidb
- ${tidb_group_name}-tidb.${namespace}
- ${tidb_group_name}-tidb.${namespace}.svc
- *.${tidb_group_name}-tidb
- *.${tidb_group_name}-tidb.${namespace}
- *.${tidb_group_name}-tidb.${namespace}.svc
- *.${tidb_group_name}-tidb-peer
- *.${tidb_group_name}-tidb-peer.${namespace}
- *.${tidb_group_name}-tidb-peer.${namespace}.svc
- ipAddresses 需要填写这两个 IP，根据需要可以填写其他 IP：
- 127.0.0.1
- ::1
- issuerRef 请填写上面创建的 Issuer；
- 其他属性请参考 cert-manager API。

通过执行下面的命令来创建证书：

```
kubectl apply -f tidb-server-cert.yaml
```

创建这个对象以后，cert-manager 会生成一个名字为 ${tidb_group_name}-tidb-server-secret 的 Secret 对象供 TiDB Server 使用。

4. 创建 Client 端证书。

创建一个 tidb-client-cert.yaml 文件，并输入以下内容：

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tidb_group_name}-tidb-client-secret
  namespace: ${namespace}
spec:
  secretName: ${tidb_group_name}-tidb-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-cert-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 ${cluster_name} 为集群的名字，${tidb_group_name} 为 TiDBGroup 的名字：

- spec.secretName 请设置为 ${tidb_group_name}-tidb-client-secret；
- usages 请添加上 client auth；
- dnsNames 和 ipAddresses 不需要填写；
- issuerRef 请填写上面创建的 Issuer；
- 其他属性请参考 cert-manager API。

通过执行下面的命令来创建证书：

```
kubectl apply -f tidb-client-cert.yaml
```

创建这个对象以后，cert-manager 会生成一个名字为 ${tidb_group_name}-tidb-client-secret 的 Secret 对象供 TiDB Client 使用。

注意：

- 由 cert-manager 签发的 Secret 中包含的 `ca.crt` 是该证书的签发 CA，并非用于验证对端 mTLS 证书的 CA。
- 本示例中，客户端和服务端的 TLS 证书由同一个 CA 签发，因此可以直接使用。如果客户端和服务端的证书由不同 CA 签发，建议使用 Trust manager 分发对应的 `ca.crt`。

7.1.1.2 第二步：部署 TiDBGroup

以下配置示例展示如何创建一个启用了 MySQL TLS 的 `TiDBGroup`：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: tidb
spec:
  cluster:
    name: tls
  version: v8.5.2
  replicas: 1
  template:
    spec:
      security:
        tls:
          mysql:
            enabled: true
      config: |
        [security]
        cluster-verify-cn = ["TiDB"]
```

7.1.1.3 第三步：配置 MySQL 客户端使用加密连接

可以根据官网文档提示，使用上面创建的 Client 证书，通过下面的方法连接 TiDB 集群：

获取 Client 证书的方式并连接 TiDB Server 的方法是：

```
kubectl get secret -n ${namespace} ${tidb_group_name}-tidb-client-secret -
    ↪ ojsonpath='{.data.tls\.crt}' | base64 --decode > client-tls.crt
kubectl get secret -n ${namespace} ${tidb_group_name}-tidb-client-secret -
    ↪ ojsonpath='{.data.tls\.key}' | base64 --decode > client-tls.key
kubectl get secret -n ${namespace} ${tidb_group_name}-tidb-client-secret -
    ↪ ojsonpath='{.data.ca\.crt}' | base64 --decode > client-ca.crt
```

```
mysql --comments -uroot -p -P 4000 -h ${tidb_host} --ssl-cert=client-tls.crt
    ↪  --ssl-key=client-tls.key --ssl-ca=client-ca.crt
```

最后请参考官网文档来验证是否正确开启了 TLS。

7.1.2 为 TiDB 组件间开启 TLS

本文主要描述了在 Kubernetes 上如何为 TiDB 集群组件间开启 TLS。开启步骤为：

1. 为即将被创建的 TiDB 集群的每个组件 Group 生成证书：
   为每个组件 Group 分别创建一套证书，保存为 Kubernetes Secret 对象：`${group_name }-${component_name}-cluster-secret`。

> 注意：
> 创建的 Secret 对象必须符合上述命名规范，否则将导致各组件部署失败。

2. 部署集群，为 Cluster Custom Resource (CR) 设置 `.spec.tlsCluster.enabled` 属性为 `true`；

> 注意：
> 在集群创建后，不能修改此字段，否则将导致集群升级失败，此时需要删除已有集群，并重新创建。

3. 配置 `pd-ctl` 和 `tikv-ctl` 连接集群。

其中，颁发证书的方式有多种，本文档提供两种方式，你也可以根据需要为 TiDB 集群颁发证书，这两种方式分别为：

- 使用 `cfssl` 系统颁发证书；
- 使用 `cert-manager` 系统颁发证书；

当需要更新已有 TLS 证书时，可参考更新和替换 TLS 证书。

7.1.2.1 第一步：为 TiDB 集群各个组件生成证书

7.1.2.1.1 使用 cfssl 系统颁发证书

1. 首先下载 cfssl 软件并初始化证书颁发机构：

```
mkdir -p ~/bin
curl -s -L -o ~/bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
curl -s -L -o ~/bin/cfssljson https://pkg.cfssl.org/R1.2/
    ↪ cfssljson_linux-amd64
chmod +x ~/bin/{cfssl,cfssljson}
export PATH=$PATH:~/bin

mkdir -p cfssl
cd cfssl
```

2. 生成 ca-config.json 配置文件：

> 注意：
>
> - TiDB 所有组件在进行组件间通信时，共用一套 TLS 证书来加密客户端与服务端的流量，因此，生成 CA 配置时必须同时指定 server auth 和 client auth。
> - 建议所有组件的证书均由同一个 CA 签发。

```
cat << EOF > ca-config.json
{
    "signing": {
        "default": {
            "expiry": "8760h"
        },
        "profiles": {
            "internal": {
                "expiry": "8760h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "client auth"
                ]
            }
        }
    }
}
EOF
```

3. 生成 ca-csr.json 配置文件：

```
cat << EOF > ca-csr.json
{
    "CN": "TiDB",
    "CA": {
        "expiry": "87600h"
    },
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "US",
            "L": "CA",
            "O": "PingCAP",
            "ST": "Beijing",
            "OU": "TiDB"
        }
    ]
}
EOF
```

4. 使用定义的选项生成 CA：

```
cfssl gencert -initca ca-csr.json | cfssljson -bare ca -
```

5. 生成证书。

   这里需要为每个 TiDB 集群的组件 Group 生成一套证书。

   - PD 证书

     首先生成默认的 pd.json 文件：

     ```
     cfssl print-defaults csr > pd.json
     ```

     然后编辑这个文件，修改 CN 和 hosts 属性：

     ```
     ...
         "CN": "TiDB",
         "hosts": [
           "127.0.0.1",
           "::1",
           "${pd_group_name}-pd",
           "${pd_group_name}-pd.${namespace}",
           "${pd_group_name}-pd.${namespace}.svc",
     ```

```
      "${pd_group_name}-pd-peer",
      "${pd_group_name}-pd-peer.${namespace}",
      "${pd_group_name}-pd-peer.${namespace}.svc",
      "*.${pd_group_name}-pd-peer",
      "*.${pd_group_name}-pd-peer.${namespace}",
      "*.${pd_group_name}-pd-peer.${namespace}.svc"
    ],
...
```

其中 `${pd_group_name}` 为 PDGroup 的名字，`${namespace}` 为 TiDB 集群部署的命名空间，你也可以添加自定义 `hosts`。

最后生成 PD 证书：

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
   ↪ -profile=internal pd.json | cfssljson -bare pd
```

- TiKV 证书

  首先生成默认的 `tikv.json` 文件：

```
cfssl print-defaults csr > tikv.json
```

  然后编辑这个文件，修改 `CN` 和 `hosts` 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${tikv_group_name}-tikv",
      "${tikv_group_name}-tikv.${namespace}",
      "${tikv_group_name}-tikv.${namespace}.svc",
      "${tikv_group_name}-tikv-peer",
      "${tikv_group_name}-tikv-peer.${namespace}",
      "${tikv_group_name}-tikv-peer.${namespace}.svc",
      "*.${tikv_group_name}-tikv-peer",
      "*.${tikv_group_name}-tikv-peer.${namespace}",
      "*.${tikv_group_name}-tikv-peer.${namespace}.svc"
    ],
...
```

  其中 `${tikv_group_name}` 为 TiKVGroup 的名字，`${namespace}` 为 TiDB 集群部署的命名空间，你也可以添加自定义 `hosts`。

  最后生成 TiKV 证书：

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
   ↪ -profile=internal tikv.json | cfssljson -bare tikv
```

- TiDB 证书

  首先生成默认的 tidb.json 文件：

```
cfssl print-defaults csr > tidb.json
```

  然后编辑这个文件，修改 CN 和 hosts 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${tidb_group_name}-tidb",
      "${tidb_group_name}-tidb.${namespace}",
      "${tidb_group_name}-tidb.${namespace}.svc",
      "${tidb_group_name}-tidb-peer",
      "${tidb_group_name}-tidb-peer.${namespace}",
      "${tidb_group_name}-tidb-peer.${namespace}.svc",
      "*.${tidb_group_name}-tidb-peer",
      "*.${tidb_group_name}-tidb-peer.${namespace}",
      "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
    ],
...
```

  其中 ${tidb_group_name} 为 TiDBGroup 的名字，${namespace} 为 TiDB 集群部署的命名空间，你也可以添加自定义 hosts。

  最后生成 TiDB 证书：

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
    ↪ -profile=internal tidb.json | cfssljson -bare tidb
```

- 其他组件证书

  除了 PD、TiKV 和 TiDB 外，其他组件的 Group 也需要生成各自的 TLS 证书。以下示例展示了生成组件证书的基本步骤：

  首先生成默认的 ${component_name}.json 文件：

```
cfssl print-defaults csr > ${component_name}.json
```

  然后编辑这个文件，修改 CN 和 hosts 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${group_name}-${component_name}",
      "${group_name}-${component_name}.${namespace}",
      "${group_name}-${component_name}.${namespace}.svc",
```

```
      "${group_name}-${component_name}-peer",
      "${group_name}-${component_name}-peer.${namespace}",
      "${group_name}-${component_name}-peer.${namespace}.svc",
      "*.${group_name}-${component_name}-peer",
      "*.${group_name}-${component_name}-peer.${namespace}",
      "*.${group_name}-${component_name}-peer.${namespace}.svc"
    ],
...
```

其中：

- ${group_name} 为组件 Group 的名字
- ${component_name} 为组件名（需使用小写字母，如 pd、tikv、tidb）
- ${namespace} 为 TiDB 集群部署的命名空间
- 你也可以添加自定义 hosts

最后生成组件证书：

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
    ↪ -profile=internal ${component_name}.json | cfssljson -bare $
    ↪ {component_name}
```

6. 创建 Kubernetes Secret 对象。

   假设你已经按照上述文档为每个组件创建了一套 Server 端证书，并为各个客户端创建了一套 Client 端证书。通过下面的命令为 TiDB 集群创建这些 Secret 对象：

   PD 集群证书 Secret：

```
kubectl create secret generic ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=pd.pem --from-file=tls
    ↪ .key=pd-key.pem --from-file=ca.crt=ca.pem
```

   TiKV 集群证书 Secret：

```
kubectl create secret generic ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tikv.pem --from-file=
    ↪ tls.key=tikv-key.pem --from-file=ca.crt=ca.pem
```

   TiDB 集群证书 Secret：

```
kubectl create secret generic ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tidb.pem --from-file=
    ↪ tls.key=tidb-key.pem --from-file=ca.crt=ca.pem
```

   其他组件证书 Secret：

```
kubectl create secret generic ${group_name}-${component_name}-cluster-
    ↪ secret --namespace=${namespace} --from-file=tls.crt=${
    ↪ component_name}.pem --from-file=tls.key=${component_name}-key.pem
    ↪  --from-file=ca.crt=ca.pem
```

这里给 PD、TiKV、TiDB 的 Server 端证书分别创建了一个 Secret 供他们启动时加载使用，另外一套 Client 端证书供他们的客户端连接使用。

7.1.2.1.2 使用 cert-manager 系统颁发证书

1. 安装 cert-manager。

   请参考官网安装：cert-manager installation on Kubernetes。

2. 创建一个 Issuer 用于给 TiDB 集群颁发证书。

   为了配置 cert-manager 颁发证书，必须先创建 Issuer 资源。

   首先创建一个目录保存 cert-manager 创建证书所需文件：

```
mkdir -p cert-manager
cd cert-manager
```

   然后创建一个 tidb-cluster-issuer.yaml 文件，输入以下内容：

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ${cluster_name}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${cluster_name}-ca
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-ca-secret
  commonName: "TiDB"
  isCA: true
  duration: 87600h # 10yrs
  renewBefore: 720h # 30d
  issuerRef:
    name: ${cluster_name}-selfsigned-ca-issuer
    kind: Issuer
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ${cluster_name}-certs-issuer
  namespace: ${namespace}
```

```
spec:
  ca:
    secretName: ${cluster_name}-ca-secret
```

其中 `${cluster_name}` 为集群的名字，上面的文件创建三个对象：

- 一个 SelfSigned 类型的 Issuer 对象（用于生成 CA 类型 Issuer 所需要的 CA 证书）
- 一个 Certificate 对象，`isCa` 属性设置为 `true`
- 一个可以用于颁发 TiDB 组件间 TLS 证书的 Issuer

最后执行下面的命令进行创建：

```
kubectl apply -f tidb-cluster-issuer.yaml
```

3. 创建组件证书。

   在 `cert-manager` 中，Certificate 资源表示证书接口，该证书将由上面创建的 Issuer 颁发并保持更新。

   根据为 TiDB 组件间通信开启加密传输文档，需要为每个组件创建一个组件证书。

   - PD 组件的证书。

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${pd_group_name}-pd-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${pd_group_name}-pd-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${pd_group_name}-pd"
  - "${pd_group_name}-pd.${namespace}"
  - "${pd_group_name}-pd.${namespace}.svc"
  - "${pd_group_name}-pd-peer"
  - "${pd_group_name}-pd-peer.${namespace}"
  - "${pd_group_name}-pd-peer.${namespace}.svc"
```

```
  - "*.${pd_group_name}-pd-peer"
  - "*.${pd_group_name}-pd-peer.${namespace}"
  - "*.${pd_group_name}-pd-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-certs-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 ${pd_group_name} 为 PDGroup 的名字，${cluster_name} 为 TiDB 集群的名字：

- spec.secretName 请设置为 ${pd_group_name}-pd-cluster-secret；
- usages 请添加上 server auth 和 client auth；
- dnsNames 需要填写这些 DNS，根据需要可以填写其他 DNS：
- ${pd_group_name}-pd
- ${pd_group_name}-pd.${namespace}
- ${pd_group_name}-pd.${namespace}.svc
- ${pd_group_name}-pd-peer
- ${pd_group_name}-pd-peer.${namespace}
- ${pd_group_name}-pd-peer.${namespace}.svc
- *.${pd_group_name}-pd-peer
- *.${pd_group_name}-pd-peer.${namespace}
- *.${pd_group_name}-pd-peer.${namespace}.svc
- ipAddresses 需要填写这两个 IP，根据需要可以填写其他 IP：
- 127.0.0.1
- ::1
- issuerRef 请填写上面创建的 Issuer；
- 其他属性请参考 cert-manager API。

创建这个对象以后，cert-manager 会生成一个名字为 ${pd_group_name}-pd-cluster-secret 的 Secret 对象供 TiDB 集群的 PD 组件使用。

- TiKV 组件的证书。

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tikv_group_name}-tikv-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${tikv_group_name}-tikv-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
```

```
  - PingCAP
commonName: "TiDB"
usages:
  - server auth
  - client auth
dnsNames:
- "${tikv_group_name}-tikv"
- "${tikv_group_name}-tikv.${namespace}"
- "${tikv_group_name}-tikv.${namespace}.svc"
- "${tikv_group_name}-tikv-peer"
- "${tikv_group_name}-tikv-peer.${namespace}"
- "${tikv_group_name}-tikv-peer.${namespace}.svc"
- "*.${tikv_group_name}-tikv-peer"
- "*.${tikv_group_name}-tikv-peer.${namespace}"
- "*.${tikv_group_name}-tikv-peer.${namespace}.svc"
ipAddresses:
- 127.0.0.1
- ::1
issuerRef:
  name: ${cluster_name}-certs-issuer
  kind: Issuer
  group: cert-manager.io
```

其中 `${tikv_group_name}` 为 TiKVGroup 的名字，`${cluster_name}` 为 TiDB 集群的名字：

- spec.secretName 请设置为 `${tikv_group_name}-tikv-cluster-secret`；
- usages 请添加上 `server auth` 和 `client auth`；
- dnsNames 需要填写这些 DNS，根据需要可以填写其他 DNS：
- `${tikv_group_name}-tikv`
- `${tikv_group_name}-tikv.${namespace}`
- `${tikv_group_name}-tikv.${namespace}.svc`
- `${tikv_group_name}-tikv-peer`
- `${tikv_group_name}-tikv-peer.${namespace}`
- `${tikv_group_name}-tikv-peer.${namespace}.svc`
- `*.${tikv_group_name}-tikv-peer`
- `*.${tikv_group_name}-tikv-peer.${namespace}`
- `*.${tikv_group_name}-tikv-peer.${namespace}.svc`
- ipAddresses 需要填写这两个 IP，根据需要可以填写其他 IP：
- 127.0.0.1
- ::1
- issuerRef 请填写上面创建的 Issuer；
- 其他属性请参考 cert-manager API。

创建这个对象以后，cert-manager 会生成一个名字为 `${tikv_group_name}-tikv-cluster-secret` 的 Secret 对象供 TiDB 集群的 TiKV 组件使用。

- TiDB 组件的证书。

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tidb_group_name}-tidb-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${tidb_group_name}-tidb-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${tidb_group_name}-tidb"
  - "${tidb_group_name}-tidb.${namespace}"
  - "${tidb_group_name}-tidb.${namespace}.svc"
  - "${tidb_group_name}-tidb-peer"
  - "${tidb_group_name}-tidb-peer.${namespace}"
  - "${tidb_group_name}-tidb-peer.${namespace}.svc"
  - "*.${tidb_group_name}-tidb-peer"
  - "*.${tidb_group_name}-tidb-peer.${namespace}"
  - "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-certs-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 ${tidb_group_name} 为 TiDBGroup 的名字，${cluster_name} 为 TiDB 集群的名字：

  - spec.secretName 请设置为 ${tidb_group_name}-tidb-cluster-secret；
  - usages 请添加上 server auth 和 client auth；
  - dnsNames 需要填写这些 DNS，根据需要可以填写其他 DNS：
  - ${tidb_group_name}-tidb
  - ${tidb_group_name}-tidb.${namespace}
  - ${tidb_group_name}-tidb.${namespace}.svc
  - ${tidb_group_name}-tidb-peer

  - `${tidb_group_name}-tidb-peer.${namespace}`
  - `${tidb_group_name}-tidb-peer.${namespace}.svc`
  - `*.${tidb_group_name}-tidb-peer`
  - `*.${tidb_group_name}-tidb-peer.${namespace}`
  - `*.${tidb_group_name}-tidb-peer.${namespace}.svc`
  - `ipAddresses` 需要填写这两个 IP，根据需要可以填写其他 IP：
  - `127.0.0.1`
  - `::1`
  - `issuerRef` 请填写上面创建的 Issuer；
  - 其他属性请参考 cert-manager API。

  创建这个对象以后，`cert-manager` 会生成一个名字为 `${tidb_group_name}-tidb-cluster-secret` 的 Secret 对象供 TiDB 集群的 TiDB 组件使用。

- 其他组件的证书。

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${group_name}-${component_name}-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${group_name}-${component_name}-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${group_name}-${component_name}"
  - "${group_name}-${component_name}.${namespace}"
  - "${group_name}-${component_name}.${namespace}.svc"
  - "${group_name}-${component_name}-peer"
  - "${group_name}-${component_name}-peer.${namespace}"
  - "${group_name}-${component_name}-peer.${namespace}.svc"
  - "*.${group_name}-${component_name}-peer"
  - "*.${group_name}-${component_name}-peer.${namespace}"
  - "*.${group_name}-${component_name}-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-certs-issuer
```

```
  kind: Issuer
  group: cert-manager.io
```

其中 ${group_name} 为组件 Group 的名字，${component_name} 为组件名，${cluster_name} 为 TiDB 集群的名字：

- spec.secretName 请设置为 ${group_name}-${component_name}-cluster-secret；
- usages 请添加上 server auth 和 client auth；
- dnsNames 需要填写这些 DNS，根据需要可以填写其他 DNS：
- ${group_name}-${component_name}
- ${group_name}-${component_name}.${namespace}
- ${group_name}-${component_name}.${namespace}.svc
- ${group_name}-${component_name}-peer
- ${group_name}-${component_name}-peer.${namespace}
- ${group_name}-${component_name}-peer.${namespace}.svc
- *.${group_name}-${component_name}-peer
- *.${group_name}-${component_name}-peer.${namespace}
- *.${group_name}-${component_name}-peer.${namespace}.svc
- ipAddresses 需要填写这两个 IP ，根据需要可以填写其他 IP：
- 127.0.0.1
- ::1
- issuerRef 请填写上面创建的 Issuer；
- 其他属性请参考 cert-manager API。

创建这个对象以后，cert-manager 会生成一个名字为 ${group_name}-${component_name}-cluster-secret 的 Secret 对象供 TiDB 集群的组件使用。

7.1.2.2 第二步：部署 TiDB 集群

在部署 TiDB 集群时，可以开启集群间的 TLS，同时可以设置 cert-allowed-cn 配置项 (TiDB 为 cluster-verify-cn)，用来验证集群间各组件证书的 CN (Common Name)。

> 注意：
>
> - 对于 TiDB v8.3.0 及之前版本，PD 的 cert-allowed-cn 配置项只能设置一个值。因此所有认证对象的 Common Name 必须设置成同一个值。
> - 从 TiDB v8.4.0 起，PD 的 cert-allowed-cn 配置项支持设置多个值。你可以根据需要在 TiDB 的 cluster-verify-cn 配置项以及其它组件的 cert-allowed-cn 配置项中设置多个 Common Name。
> - 详情参考为 TiDB 组件间通信开启加密传输。

按照如下步骤部署 TiDB 集群并开启集群间的 TLS：

创建 tidb-cluster.yaml 文件：

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  tlsCluster:
    enabled: true
---
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: ${pd_group_name}
  namespace: ${namespace}
spec:
  cluster:
    name: ${cluster_name}
  version: v8.5.2
  replicas: 3
  template:
    spec:
      config: |
        [security]
        cert-allowed-cn = ["TiDB"]
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 20Gi
---
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: ${tikv_group_name}
  namespace: ${namespace}
spec:
  cluster:
    name: ${cluster_name}
  version: v8.5.2
  replicas: 3
  template:
    spec:
      config: |
```

```
      [security]
      cert-allowed-cn = ["TiDB"]
    volumes:
    - name: data
      mounts:
      - type: data
      storage: 100Gi
---
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: ${tidb_group_name}
  namespace: ${namespace}
spec:
  cluster:
    name: ${cluster_name}
  version: v8.5.2
  replicas: 1
  template:
    spec:
      config: |
        [security]
        cluster-verify-cn = ["TiDB"]
```

然后使用 `kubectl apply -f tidb-cluster.yaml` 来创建 TiDB 集群。

7.1.3 以非 root 用户运行容器

在某些 Kubernetes 环境中，容器无法以 root 用户身份运行。出于安全考虑，建议在生产环境中以非 root 用户运行容器，以降低潜在攻击带来的安全风险。本文介绍如何通过配置 securityContext 实现以非 root 用户运行容器。

7.1.3.1 配置 TiDB Operator 相关的容器

对于 TiDB Operator 相关的容器，可以在 Helm 的 `values.yaml` 文件中配置安全上下文 (`securityContext`)。

以下是一个配置示例：

```
controllerManager:
  securityContext:
    runAsUser: 1000
    runAsGroup: 2000
    fsGroup: 2000
```

7.1.3.2 配置按照 CR 生成的容器

对于按照 Custom Resource (CR) 生成的容器，可以在任意一种 CR（例如 PDGroup、TiDBGroup、TiKVGroup、TiFlashGroup、TiCDCGroup、Backup、CompactBackup、BackupSchedule、Restore）中配置安全上下文 (securityContext)。

- 对于 PDGroup、TiDBGroup、TiKVGroup、TiFlashGroup、TiCDCGroup 等 CR，可以通过 Overlay 的方式配置安全上下文。配置 PDGroup CR 的示例如下：

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            securityContext:
              runAsUser: 1000
              runAsGroup: 2000
              fsGroup: 2000
```

- 对于 Backup、CompactBackup、BackupSchedule、Restore 等 CR，可以在 spec 中配置 podSecurityContext。配置 Backup CR 的示例如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup
spec:
  podSecurityContext:
    runAsUser: 1000
    runAsGroup: 2000
    fsGroup: 2000
```

7.1.4 更新和替换 TLS 证书

本文以更新和替换 TiDB 集群中 PD、TiKV、TiDB 组件间的 TLS 证书为例，介绍在证书过期之前，如何更新和替换相应组件的证书。

如需要更新和替换集群中其他组件间的证书、TiDB Server 端证书或 MySQL Client 端证书，可使用类似的步骤进行操作。

本文的更新和替换操作假定原证书尚未过期。若原证书已经过期或失效，可参考为 TiDB 组件间开启 TLS 或为 MySQL 客户端开启 TLS 生成新的证书并重启集群。

7.1.4.1 更新和替换 cfssl 系统颁发的证书

如原 TLS 证书是使用 cfssl 系统颁发的证书，且原证书尚未过期，可按如下步骤更新和替换 PD、TiKV、TiDB 组件间的证书。

7.1.4.1.1 更新和替换 CA 证书

> **注意：**
> 若无需更新 CA 证书，可跳过本节中的操作，直接按更新和替换组件间证书进行操作。

1. 备份原 CA 证书与密钥。

```
mv ca.pem ca.old.pem && \
mv ca-key.pem ca-key.old.pem
```

2. 基于原 CA 证书的配置与证书签名请求 (CSR)，生成新的 CA 证书和密钥。

```
cfssl gencert -initca ca-csr.json | cfssljson -bare ca -
```

> **注意：**
> 配置文件与 CSR 中的 expiry 如有需要可进行更新。

3. 备份新 CA 证书与密钥，并基于原有 CA 证书及新的 CA 证书生成组合 CA 证书。

```
mv ca.pem ca.new.pem && \
mv ca-key.pem ca-key.new.pem && \
cat ca.new.pem ca.old.pem > ca.pem
```

4. 基于组合 CA 证书更新各相应的 Kubernetes Secret 对象。

```
kubectl create secret generic ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=pd-server.pem --from-
    ↪ file=tls.key=pd-server-key.pem --from-file=ca.crt=ca.pem --dry-
    ↪ run=client -o yaml | kubectl apply -f -
kubectl create secret generic ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tikv-server.pem --from
    ↪ -file=tls.key=tikv-server-key.pem --from-file=ca.crt=ca.pem --dry
    ↪ -run=client -o yaml | kubectl apply -f -
```

```
kubectl create secret generic ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tidb-server.pem --from
    ↪ -file=tls.key=tidb-server-key.pem --from-file=ca.crt=ca.pem --dry
    ↪ -run=client -o yaml | kubectl apply -f -
```

其中 ${pd_group_name}、${tikv_group_name}、${tidb_group_name} 为组件 Group 的名字，${namespace} 为 TiDB 集群部署的命名空间。

> **注意：**
> 上述示例命令中仅更新了 PD、TiKV、TiDB 的组件间 Server 端 CA 证书与 Client 端 CA 证书，如需更新其他如 TiCDC、TiFlash、TiProxy 等的 Server 端 CA 证书，可使用类似命令进行更新。

5. 参考滚动重启 TiDB 集群对需要加载组合 CA 证书的组件进行滚动重启。

   滚动重启完成后，基于组合 CA 证书，各组件将能同时接受由原 CA 证书与新 CA 证书签发的证书。

7.1.4.1.2 更新和替换组件间证书

> **注意：**
> 在更新和替换组件间证书前，请确保更新前后的组件间证书均能被 CA 证书验证为有效。如已更新和替换 CA 证书，请确保 TiDB 集群已基于新的 CA 证书完成重启。

1. 基于各组件原配置信息，生成新的 Server 端与 Client 端证书。

```
cfssl gencert -ca=ca.new.pem -ca-key=ca-key.new.pem -config=ca-config.
    ↪ json -profile=internal pd-server.json | cfssljson -bare pd-server
cfssl gencert -ca=ca.new.pem -ca-key=ca-key.new.pem -config=ca-config.
    ↪ json -profile=internal tikv-server.json | cfssljson -bare tikv-
    ↪ server
cfssl gencert -ca=ca.new.pem -ca-key=ca-key.new.pem -config=ca-config.
    ↪ json -profile=internal tidb-server.json | cfssljson -bare tidb-
    ↪ server
```

> **注意：**

- 上述示例命令中假定已参考更新和替换 CA 证书中的步骤备份了新的 CA 证书与密钥为 ca.new.pem 与 ca-key.new.pem。如未更新 CA 证书与密钥，请修改示例命令中的对应参数为 ca.pem 与 ca-key.pem。
- 上述示例命令中仅生成了 PD、TiKV、TiDB 的组件间 Server 端证书与 Client 端证书，如需生成其他如 TiCDC、TiFlash 等的 Server 端证书，可使用类似命令进行生成。

2. 基于新生成的 Server 端与 Client 端证书，更新相应的 Kubernetes Secret 对象。

```
kubectl create secret generic ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=pd-server.pem --from-
    ↪ file=tls.key=pd-server-key.pem --from-file=ca.crt=ca.pem --dry-
    ↪ run=client -o yaml | kubectl apply -f -
kubectl create secret generic ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tikv-server.pem --from
    ↪ -file=tls.key=tikv-server-key.pem --from-file=ca.crt=ca.pem --dry
    ↪ -run=client -o yaml | kubectl apply -f -
kubectl create secret generic ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tidb-server.pem --from
    ↪ -file=tls.key=tidb-server-key.pem --from-file=ca.crt=ca.pem --dry
    ↪ -run=client -o yaml | kubectl apply -f -
```

其中 ${pd_group_name}、${tikv_group_name}、${tidb_group_name} 为组件 Group 的名字，${namespace} 为 TiDB 集群部署的命名空间。

注意：
上述示例命令中仅更新了 PD、TiKV、TiDB 的组件间 Server 端证书与 Client 端证书，如需更新其他如 TiCDC、TiFlash、TiProxy 等的 Server 端证书，可使用类似命令进行更新。

3. 参考滚动重启 TiDB 集群对需要加载新证书的组件进行滚动重启。

   滚动重启完成后，各组件将使用新的证书进行 TLS 通信。如有参考更新和替换 CA 证书并使各组件加载了组合 CA 证书，则其仍能接受由原 CA 证书签发的证书。

7.1.4.1.3 可选：移除组合 CA 证书中的原 CA 证书

若同时参考更新和替换 CA 证书与更新和替换组件间证书更新与替换了组合 CA 证书与 Server 端、Client 端组件证书，且计划移除原 CA 证书（如原 CA 证书已过期或原 CA 证书的密钥被盗），则可按如下步骤移除原 CA 证书。

1. 基于新 CA 证书更新 Kubernetes Secret 对象。

```
kubectl create secret generic ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=pd-server.pem --from-
    ↪ file=tls.key=pd-server-key.pem --from-file=ca.crt=ca.new.pem --
    ↪ dry-run=client -o yaml | kubectl apply -f -
kubectl create secret generic ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tikv-server.pem --from
    ↪ -file=tls.key=tikv-server-key.pem --from-file=ca.crt=ca.new.pem
    ↪ --dry-run=client -o yaml | kubectl apply -f -
kubectl create secret generic ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tidb-server.pem --from
    ↪ -file=tls.key=tidb-server-key.pem --from-file=ca.crt=ca.new.pem
    ↪ --dry-run=client -o yaml | kubectl apply -f -
```

其中 `${pd_group_name}`、`${tikv_group_name}`、`${tidb_group_name}` 为组件 Group 的名字，`${namespace}` 为 TiDB 集群部署的命名空间。

> **注意：**
> 上述示例命令中假定已参考更新和替换 CA 证书中的步骤备份了新的 CA 证书为 `ca.new.pem`

2. 参考滚动重启 TiDB 集群对需要加载新证书的组件进行滚动重启。

   滚动重启完成后，各组件将仅能接受由新 CA 证书签发的证书。

7.1.4.2 更新和替换 `cert-manager` 颁发的证书

如原 TLS 证书是使用 `cert-manager` 系统颁发的证书，且原证书尚未过期，根据是否需要更新 CA 证书需要分别处理。

7.1.4.2.1 更新和替换 CA 证书

7.1.4.2.2 仅更新和替换组件间证书

使用 cert-manager 颁发证书时，可通过配置 `Certificate` 资源的 `spec.renewBefore` 字段，让 cert-manager 在证书过期前自动进行更新。

1. cert-manager 支持在证书过期前自动更新各组件的证书及对应的 Kubernetes Secret 对象。如需手动更新，可以参考使用 cmctl renew 证书。
2. 对于各组件间的证书，各组件会在之后新建连接时自动重新加载新的证书，无需手动操作。

注意：

- 各组件目前暂不支持 CA 证书的自动重新加载，需要参考更新和替换 CA 证书进行处理。
- 对于 TiDB Server 端证书，可参考以下任意方式进行手动重加载：
  - 参考重加载证书、密钥和 CA。
  - 参考滚动重启 TiDB 集群对 TiDB Server 进行滚动重启。

7.2 手动扩缩容 Kubernetes 上的 TiDB 集群

本文介绍如何对部署在 Kubernetes 上的 TiDB 集群进行手动水平扩缩容和垂直扩缩容。

7.2.1 水平扩缩容

水平扩缩容操作是指通过增加或减少组件的 Pod 的数量，来达到集群扩缩容的目的。可通过修改组件的 replicas 参数来控制 Pod 数量，从而实现扩容或缩容。

- 如果要进行扩容操作，可将某个组件的 replicas 值调大。扩容操作会增加组件 Pod，直到 Pod 数量与 replicas 值相等。
- 如果要进行缩容操作，可将某个组件的 replicas 值调小。缩容操作会删除组件 Pod，直到 Pod 数量与 replicas 值相等。

如果要对 TiDB 集群进行水平扩缩容，你可以使用 kubectl 修改对应组件的 Component Group Custom Resource (CR) 对象中的 spec.replicas 至期望值。

1. 按需修改 TiDB 集群组件的 replicas 值。例如，执行以下命令可将 PD 的 replicas 值设置为 3：

```
kubectl patch -n ${namespace} pdgroup ${name} --type merge --patch '{"
    ↪ spec":{"replicas":3}}'
```

2. 查看 Kubernetes 集群中对应组件的 Component Group CR 是否更新为期望的配置。例如，执行以下命令查看 PDGroup CR：

```
kubectl get pdgroup ${name} -n ${namespace}
```

上述命令输出的 DESIRED 的值预期应与你之前配置的值一致。

3. 观察 Pod 是否新增或者减少：

```
kubectl -n ${namespace} get pod -w
```

当所有组件的 Pod 数量都达到了预设值，并且都进入 Running 状态后，水平扩缩容完成。

PD 和 TiDB 通常需要 10 到 30 秒左右的时间进行扩容或者缩容。

TiKV 组件由于涉及到数据搬迁，通常需要 3 到 5 分钟来进行扩容或者缩容。

注意：

- TiKV 组件在缩容过程中，TiDB Operator 会调用 PD 接口将对应 TiKV 标记为下线，然后将其上数据迁移到其它 TiKV 节点，在数据迁移期间 TiKV Pod 依然是 Running 状态，数据迁移完成后对应 Pod 才会被删除，缩容时间与待缩容的 TiKV 上的数据量有关，可以通过 `kubectl get -n ${namespace} tikv` 查看 TiKV 是否处于下线 Removing 状态。
- 当 Serving 状态的 TiKV 数量小于或等于 PD 配置中 MaxReplicas 的参数值时，无法缩容 TiKV 组件。
- TiKV 组件不支持在缩容过程中进行扩容操作，强制执行此操作可能导致集群状态异常。
- TiFlash 组件缩容处理逻辑和 TiKV 组件相同。

7.2.2 垂直扩缩容

垂直扩缩容操作指的是通过增加或减少 Pod 的资源限制，来达到集群扩缩容的目的。垂直扩缩容本质上是 Pod 滚动升级的过程。

如果要对 PD、TiKV、TiDB、TiProxy、TiFlash 或 TiCDC 进行垂直扩缩容，通过 kubectl 修改对应的 Component Group CR 对象的 `spec.template.spec.resources` 至期望值。

注意：

暂不支持原地调整 Pod 的资源。

7.2.2.1 查看垂直扩缩容进度

```
kubectl -n ${namespace} get pod -w
```

当所有 Pod 都重建完毕进入 Running 状态后，垂直扩缩容完成。

注意：

- 如果在垂直扩容时修改了资源的 `requests` 字段，并且 PD、TiKV、TiFlash、TiCDC 使用了 `Local PV`，那升级后 Pod 还会调度回原节点，如果原节点资源不够，则会导致 Pod 一直处于 `Pending` 状态而影响服务。
- TiDB 是一个可水平扩展的数据库，推荐通过增加节点个数发挥 TiDB 集群可水平扩展的优势，而不是类似传统数据库升级节点硬件配置来实现垂直扩容。

7.2.3 扩缩容故障诊断

无论是水平扩缩容、或者是垂直扩缩容，都可能遇到资源不够时造成 Pod 出现 Pending 的情况。可以参考Pod 处于 Pending 状态来进行处理。

7.3 升级

7.3.1 升级 TiDB Operator

本文介绍如何升级 TiDB Operator 到指定版本。

7.3.1.1 升级注意事项

暂不支持从 v1.x 升级到 v2.x 版本。

7.3.1.2 升级 CRD

执行以下命令升级 TiDB Operator 的 Custom Resource Definition (CRD)。请将 `${version}` 替换为目标 TiDB Operator 版本，例如 `v2.0.0-beta.0`：

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ ${version}/tidb-operator.crds.yaml --server-side
```

7.3.1.3 升级 TiDB Operator 组件

你可以使用以下两种方式升级 TiDB Operator 组件：

- 方式一：使用 `kubectl apply`
- 方式二：使用 Helm

7.3.1.3.1 方式一：使用 kubectl apply 升级

执行以下命令升级 TiDB Operator 组件：

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ ${version}/tidb-operator.yaml --server-side
```

此命令会升级部署在 tidb-admin 命名空间中的 TiDB Operator。你可以运行以下命令确认 Pod 是否升级成功：

```
kubectl get pods -n tidb-admin
```

预期输出示例：

```
NAME                              READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m
```

7.3.1.3.2 方式二：使用 Helm 升级

如果你使用 Helm 部署 TiDB Operator，可以使用 helm upgrade 命令进行升级。

执行以下命令升级 TiDB Operator：

```
helm upgrade tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator --
    ↪ version=${version} --namespace=tidb-admin
```

命令说明：

- tidb-operator：TiDB Operator 的 Helm release 名称。如果你的 release 名称不同，请替换为实际的名称。
- ${version}：目标升级的 TiDB Operator 版本号，例如 v2.0.0-beta.0。
- --namespace=tidb-admin：指定 TiDB Operator 所在的命名空间。如果你的命名空间不同，请替换为实际的命名空间。

升级完成后，你可以通过以下命令检查 Pod 状态，确认升级是否成功：

```
kubectl get pods -n tidb-admin
```

使用自定义配置升级

如果在部署或之前的升级中使用了自定义配置（即修改了 values.yaml 文件），请确保在本次升级中也使用这些自定义配置。具体升级步骤如下：

1. 获取当前部署所使用的 values.yaml 文件：

   ```
   helm get values tidb-operator -n tidb-admin > values.yaml
   ```

2. 获取目标版本的默认配置文件 values-new.yaml：

```
helm show values oci://ghcr.io/pingcap/charts/tidb-operator --version=$
    ↪ {version} > values-new.yaml
```

3. 对比 values.yaml 和 values-new.yaml 两个文件，将你的自定义配置项合并到 values-new.yaml 中。

4. 使用合并后的 values-new.yaml 文件进行升级：

```
helm upgrade tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator
    ↪ --version=${version} -f values-new.yaml --namespace=tidb-admin
```

7.3.2 升级 Kubernetes 上的 TiDB 集群

如果你使用 TiDB Operator 部署管理 Kubernetes 上的 TiDB 集群，可以通过滚动更新来升级 TiDB 集群的版本，减少对业务的影响。本文介绍如何使用滚动更新来升级 Kubernetes 上的 TiDB 集群。

7.3.2.1 滚动更新功能介绍

Kubernetes 提供了滚动更新功能，在不影响应用可用性的前提下执行更新。

使用滚动更新时，TiDB Operator 会等待新版本的 Pod 正常运行后，再处理下一个 Pod。

滚动更新中，TiDB Operator 会自动处理 PD 和 TiKV 的 Leader 迁移。因此，在多节点的部署拓扑下（最小环境：PD * 3、TiKV * 3、TiDB * 2），滚动更新 TiKV、PD 不会影响业务正常运行。对于有连接重试功能的客户端，滚动更新 TiDB 同样不会影响业务。

警告：

- 对于无法进行连接重试的客户端，滚动更新 TiDB 会导致连接到被关闭节点的数据库的连接失效，造成部分业务请求失败。对于这类业务，推荐在客户端添加重试功能，或者在低峰期进行 TiDB 的滚动更新操作。
- 升级前，请执行 ADMIN SHOW DDL [JOBS|JOB QUERIES]，确认没有正在进行的 DDL 操作。

7.3.2.2 升级前准备

1. 查阅升级兼容性说明，了解升级的注意事项。注意包括补丁版本在内的所有 TiDB 版本目前暂不支持降级或升级后回退。

2. 查阅 TiDB release notes 中各中间版本的兼容性变更。如果有任何变更影响到了你的升级，请采取相应的措施。

   例如，从 TiDB v6.4.0 升级至 v6.5.2 时，需查阅以下各版本的兼容性变更：

   - TiDB v6.5.0 release notes 中的兼容性变更和废弃功能
   - TiDB v6.5.1 release notes 中的兼容性变更
   - TiDB v6.5.2 release notes 中的兼容性变更

   如果从 v6.3.0 或之前版本升级到 v6.5.2，也需要查看中间版本 release notes 中提到的兼容性变更信息。

7.3.2.3 升级步骤

1. 修改待升级集群的各组件 Group 的版本配置，通过 `version` 字段指定每个组件的目标版本。例如：

```
spec:
  template:
    spec:
      version: v8.5.2
```

   你可以使用 `kubectl apply` 命令一次性更新所有组件的配置，也可以通过 `kubectl edit` 逐个修改组件。TiDB Operator 会自动处理升级顺序，并在组件未满足升级前置条件时阻止升级继续执行。

> **注意：**
> TiDB Operator 要求集群中的所有组件使用相同版本。请确保所有组件的 `spec.template.spec.version` 字段设置为相同的版本号。

2. 查看升级进度：

```
watch kubectl -n ${namespace} get pod -o wide
```

   当所有 Pod 都重建完毕进入 `Running` 状态后，升级完成。

7.4 备份与恢复

7.4.1 备份与恢复简介

本文档介绍如何对 Kubernetes 上的 TiDB 集群进行数据备份和数据恢复。备份与恢复中所使用的工具有 Dumpling、TiDB Lightning 和 BR。

Dumpling 是一个数据导出工具，该工具可以把存储在 TiDB/MySQL 中的数据导出为 SQL 或者 CSV 格式，可以用于完成逻辑上的全量备份或者导出。

TiDB Lightning 是一个数据导入工具，该工具可以把 Dumpling 或 CSV 输出格式的数据快速导入到 TiDB 中，可以用于完成逻辑上的全量恢复或者导入。

BR 是 TiDB 分布式备份恢复的命令行工具，用于对 TiDB 集群进行数据备份和恢复。相比 Dumpling 和 Mydumper，BR 更适合大数据量的场景，BR 只支持 TiDB v3.1 及以上版本。如果需要对延迟不敏感的增量备份，请参阅 BR。如果需要实时的增量备份，请参阅 TiCDC。

7.4.1.1 使用场景

7.4.1.1.1 数据备份

如果你对数据备份有以下要求，可考虑使用 BR 对 TiDB 进行数据备份：

- 备份的数据量较大（大于 1 TiB），而且要求备份速度较快
- 直接备份数据的 SST 文件（键值对）
- 对延迟不敏感的增量备份

BR 相关使用文档可参考：

- 使用 BR 备份 TiDB 集群到兼容 S3 的存储
- 使用 BR 备份 TiDB 集群到 GCS
- 使用 BR 备份 TiDB 集群到 Azure Blob Storage

7.4.1.1.2 数据恢复

如果你需要从由 BR 备份出的 SST 文件对 TiDB 进行数据恢复，则应使用 BR。相关使用文档可参考：

- 使用 BR 恢复兼容 S3 的存储上的备份数据
- 使用 BR 恢复 GCS 上的备份数据
- 使用 BR 恢复 Azure Blob Storage 上的备份数据

7.4.1.2 备份与恢复过程

为了对 Kubernetes 上的 TiDB 集群进行数据备份，用户需要创建一个`Backup` Custom Resource (CR) 对象来描述一次备份，或者创建一个`BackupSchedule` CR 对象来描述一个定时备份。

为了对 Kubernetes 上的 TiDB 集群进行数据恢复，用户可以通过创建一个`Restore` CR 对象来描述一次恢复。

在创建完对应的 CR 对象后，TiDB Operator 将根据相应配置并选择对应的工具执行备份或恢复。

7.4.1.3 删除备份的 Backup CR

你可以通过下述语句来删除对应的备份 CR 或定时快照备份 CR。

```
kubectl delete backup ${name} -n ${namespace}
kubectl delete backupschedule ${name} -n ${namespace}
```

如果将 spec.cleanPolicy 设置为 Delete 时，TiDB Operator 在删除 CR 时会同时清理备份文件。

当你删除 CR 时，TiDB Operator 会尝试自动停止正在运行的日志备份任务。此自动停止功能仅适用于正常运行的日志备份任务，不会处理出现错误或失败状态的任务。

在满足上述条件时，如果需要删除 namespace，建议首先删除所有的 Backup/BackupSchedule CR，再删除 namespace。

如果直接删除存在 Backup/BackupSchedule CR 的 namespace，TiDB Operator 会持续尝试创建 Job 清理备份的数据，但因为 namespace 处于 Terminating 状态而创建失败，从而导致 namespace 卡在该状态。

这时需要通过下述命令删除 finalizers：

```
kubectl patch -n ${namespace} backup ${name} --type merge -p '{"metadata":{"
    ↪ finalizers":[]}}'
```

7.4.1.3.1 清理备份文件

TiDB Operator 清理备份文件的方式为：循环删除备份文件，一次批量删除多个文件。对于每次批量删除多个文件的操作，根据备份使用的后端存储类型的不同，删除方式不同。

- S3 兼容的后端存储采用并发批量删除方式。TiDB Operator 启动多个 Go 协程，每个 Go 协程每次调用批量删除接口 DeleteObjects 来删除多个文件。
- 其他类型的后端存储采用并发删除方式。TiDB Operator 启动多个 Go 协程，每个 Go 协程每次删除一个文件。

你可以使用 Backup CR 中的以下字段控制清理行为：

- .spec.cleanOption.pageSize：指定每次批量删除的文件数量。默认值为 10000。
- .spec.cleanOption.disableBatchConcurrency：当设置为 true 时，TiDB Operator 会禁用并发批量删除方式，使用并发删除方式。

  如果 S3 兼容的后端存储不支持 DeleteObjects 接口，默认的并发批量删除会失败，需要配置该字段为 true 来使用并发删除方式。
- .spec.cleanOption.batchConcurrency：指定并发批量删除方式下启动的 Go 协程数量。默认值为 10。
- .spec.cleanOption.routineConcurrency：指定并发删除方式下启动的 Go 协程数量。默认值为 100。

7.4.2 备份与恢复 CR 介绍

本文档介绍用于备份与恢复的 Backup、CompactBackup、Restore 及 BackupSchedule 等 Custom Resource (CR) 资源的各字段，确保更好地对 Kubernetes 上的 TiDB 集群进行数据备份和数据恢复。

7.4.2.1 Backup CR 字段介绍

为了对 Kubernetes 上的 TiDB 集群进行数据备份，用户可以通过创建一个自定义的 Backup CR 对象来描述一次备份，具体备份过程可参考数据备份中列出的文档。以下介绍 Backup CR 各个字段的具体含义。

7.4.2.1.1 通用字段介绍

- `.spec.toolImage`：用于指定 Backup 使用的工具镜像。
  - 如果未指定或者为空，默认使用镜像 `pingcap/br:${tikv_version}` 进行备份。
  - 如果指定了 BR 的版本，例如 `.spec.toolImage: pingcap/br:v8.5.2`，那么使用指定的版本镜像进行备份。
  - 如果指定了镜像但未指定版本，例如 `.spec.toolImage: private/registry/br`，那么使用镜像 `private/registry/br:${tikv_version}` 进行备份。
- `.spec.backupType`：指定 Backup 类型，该字段仅在使用 BR 备份时有效，目前支持以下三种类型，可以结合 `.spec.tableFilter` 配置表库过滤规则：
  - `full`：对 TiDB 集群所有的 database 数据执行备份。
  - `db`：对 TiDB 集群一个 database 的数据执行备份。
  - `table`：对 TiDB 集群中指定表的数据执行备份。
- `.spec.backupMode`：指定 Backup 的模式，默认为 `snapshot`。目前支持以下两种类型：
  - `snapshot`：基于 KV 层的快照备份。
  - `log`：从 KV 层备份实时数据变更日志数据。
- `.spec.logSubcommand`：指定日志备份任务的子命令，用于控制日志备份任务的状态。该字段支持以下三个选项：
  - `log-start`：启动一个新的日志备份任务，或恢复一个已暂停的任务。使用此命令可以开始日志备份流程，或从暂停状态恢复任务。
  - `log-stop`：永久停止日志备份任务。执行此命令后，Backup CR 会进入停止状态，且无法再次启动。
  - `log-pause`：暂停当前正在进行的日志备份任务。暂停任务后，你可以使用 `log-start` 命令恢复任务。

- `.spec.cleanPolicy`：备份集群后删除 Backup CR 时的备份文件清理策略。如果不配置该字段，或者配置该字段的值为以下三种以外的值，均会保留备份出的文件。目前支持三种清理策略：

  - `Retain`：任何情况下，删除 Backup CR 时会保留备份出的文件。
  - `Delete`：任何情况下，删除 Backup CR 时会删除备份出的文件。
  - `OnFailure`：如果备份中失败，删除 Backup CR 时会删除备份出的文件。

- `.spec.cleanOption`：备份集群后删除 Backup CR 时的备份文件清理行为。更多说明请参阅清理备份文件。

- `.spec.storageClassName`：备份时所需的 persistent volume (PV) 类型。

- `.spec.storageSize`：备份时指定所需的 PV 大小，默认为 100 GiB。该值应大于备份 TiDB 集群数据的大小。一个 TiDB 集群的 Backup CR 对应的 PVC 名字是确定的，如果集群命名空间中已存在该 PVC 并且其大小小于 `.spec.storageSize`，这时需要先删除该 PVC 再运行 Backup job。

- `.spec.resources`：指定运行备份任务的 Pod 的资源请求与上限值。

- `.spec.env`：指定运行备份任务的 Pod 的环境变量信息。

- `.spec.affinity`：指定运行备份任务的 Pod 亲和性配置，关于 affinity 的使用说明，请参阅 Affinity & AntiAffinity。

- `.spec.tolerations`：指定运行备份任务的 Pod 能够调度到带有与之匹配的污点 (Taint) 的节点上。关于污点与容忍度的更多说明，请参阅 Taints and Tolerations。

- `.spec.podSecurityContext`：指定运行备份任务的 Pod 的安全上下文配置，允许 Pod 以非 root 用户的方式运行，关于 podSecurityContext 的更多说明，请参阅以非 root 用户运行容器。

- `.spec.priorityClassName`：指定运行备份任务的 Pod 的 priorityClass 的名称，以设置运行优先级，关于 priorityClass 的更多说明，请参阅 Pod Priority and Preemption。

- `.spec.imagePullSecrets`：指定运行备份任务的 Pod 的 imagePullSecrets。

- `.spec.serviceAccount`：备份时指定所使用的 ServiceAccount 名称。

- `.spec.useKMS`：备份时指定是否使用 AWS-KMS 解密备份使用的 S3 存储密钥。

- `.spec.tableFilter`：备份时指定让 BR 备份符合 table-filter 规则的表。默认情况下该字段可以不用配置。当不配置时，BR 会备份除系统库以外的所有数据库。

> 注意：
> 如果要使用排除规则 `"!db.table"` 导出除 `db.table` 的所有表，那么在 `"!db.table"` 前必须先添加 `*.*` 规则。如下面例子所示：

```
tableFilter:
- "*.*"
- "!db.table"
```

- .spec.backoffRetryPolicy：指定备份的 Job/Pod 发生非正常失败（如节点资源不足被 Kubernetes 杀死）时的重试策略。这个策略目前只适用于 snapshot 备份。
  - minRetryDuration：发现异常失败后的最小重试间隔，重试间隔随失败次数增加，RetryDuration = minRetryDuration << (retryNum -1)。时间设置格式参考 func ParseDuration，默认 300s。
  - maxRetryTimes：最大重试次数，默认 2。
  - retryTimeout：重试超时时间，从首次发现异常失败开始计算。时间设置格式参考 func ParseDuration，默认 30m。

7.4.2.1.2 BR 字段介绍

- .spec.br.cluster：代表需要备份的集群名字。
- .spec.br.clusterNamespace：代表需要备份的集群所在的 namespace。
- .spec.br.logLevel：代表日志的级别。默认为 info。
- .spec.br.statusAddr：为 BR 进程监听一个进程状态的 HTTP 端口，方便用户调试。如果不填，则默认不监听。
- .spec.br.concurrency：备份时每一个 TiKV 进程使用的线程数。备份时默认为 4，恢复时默认为 128。
- .spec.br.rateLimit：是否对流量进行限制。单位为 MB/s，例如设置为 4 代表限速 4 MB/s，默认不限速。
- .spec.br.checksum：是否在备份结束之后对文件进行验证。默认为 true。
- .spec.br.timeAgo：备份 timeAgo 以前的数据，默认为空（备份当前数据），支持“1.5h”，“2h45m” 等数据。
- .spec.br.sendCredToTikv：BR 进程是否将自己的 AWS 权限、Google Cloud 权限或者 Azure 权限传输给 TiKV 进程。默认为 true。
- .spec.br.onLine：restore 时是否启用在线恢复功能。
- .spec.br.options：BR 工具支持的额外参数，需要以字符串数组的形式传入。可用于指定 lastbackupts 以进行增量备份。

7.4.2.1.3 S3 存储字段介绍

- .spec.s3.provider：支持的兼容 S3 的 provider。

  更多支持的兼容 S3 的 provider 如下：
  - alibaba：Alibaba Cloud Object Storage System (OSS)，formerly Aliyun

  - `digitalocean`: Digital Ocean Spaces
  - `dreamhost`: Dreamhost DreamObjects
  - `ibmcos`: IBM COS S3
  - `minio`: Minio Object Storage
  - `netease`: Netease Object Storage (NOS)
  - `wasabi`: Wasabi Object Storage
  - `other`: Any other S3 compatible provider

- `.spec.s3.region`：使用 Amazon S3 存储备份，需要配置 Amazon S3 所在的 region。
- `.spec.s3.bucket`：兼容 S3 存储的 bucket 名字。
- `.spec.s3.prefix`：如果设置了这个字段，则会使用这个字段来拼接在远端存储的存储路径 `s3://${.spec.s3.bucket}/${.spec.s3.prefix}/backupName`。
- `.spec.s3.path`：指定备份文件在远端存储的存储路径，该字段仅在使用 Dumpling 备份或 Lightning 恢复时有效，例如 `s3://test1-demo1/backup-2019-12-11T04:32:12Z.tgz`。
- `.spec.s3.endpoint`：兼容 S3 的存储服务 endpoint，例如 `http://minio.minio.svc.cluster.local:9000`。
- `.spec.s3.secretName`：访问兼容 S3 存储的密钥信息（包含 access key 和 secret key）的 Secret 名称。
- `.spec.s3.sse`：指定 S3 的服务端加密方式，例如 `aws:kms`。
- `.spec.s3.acl`：支持的 access-control list (ACL) 策略。

  Amazon S3 支持以下几种 access-control list (ACL) 策略：

  - `private`
  - `public-read`
  - `public-read-write`
  - `authenticated-read`
  - `bucket-owner-read`
  - `bucket-owner-full-control`

  如果不设置 ACL 策略，则默认使用 `private` 策略。ACL 策略的详细介绍，参考 AWS 官方文档。

- `.spec.s3.storageClass`：支持的 `storageClass` 类型。

  Amazon S3 支持以下几种 `storageClass` 类型：

  - `STANDARD`
  - `REDUCED_REDUNDANCY`
  - `STANDARD_IA`
  - `ONEZONE_IA`

  - GLACIER
  - DEEP_ARCHIVE

  如果不设置 storageClass，则默认使用 STANDARD_IA。storageClass 的详细介绍，参考 AWS 官方文档。

7.4.2.1.4 GCS 存储字段介绍

- .spec.gcs.projectId：代表 Google Cloud 上用户项目的唯一标识。具体获取该标识的方法可参考 Google Cloud 官方文档。
- .spec.gcs.location：指定 GCS bucket 所在的区域，例如 us-west2。
- .spec.gcs.path：指定备份文件在远端存储的存储路径，该字段仅在使用 Dumpling 备份或 Lightning 恢复时有效，例如 gcs://test1-demo1/backup-2019-11-11T16:06:05Z.tgz。
- .spec.gcs.secretName：指定存储 GCS 用户账号认证信息的 Secret 名称。
- .spec.gcs.bucket：存储数据的 bucket 名字。
- .spec.gcs.prefix：如果设置了这个字段，则会使用这个字段来拼接在远端存储的存储路径 gcs://${.spec.gcs.bucket}/${.spec.gcs.prefix}/backupName。
- .spec.gcs.storageClass：GCS 支持以下几种 storageClass 类型：

  - MULTI_REGIONAL
  - REGIONAL
  - NEARLINE
  - COLDLINE
  - DURABLE_REDUCED_AVAILABILITY

  如果不设置 storageClass，则默认使用 COLDLINE。这几种存储类型的详细介绍可参考 GCS 官方文档。

- .spec.gcs.objectAcl：设置 object access-control list (ACL) 策略。

  GCS 支持以下几种 ACL 策略：

  - authenticatedRead
  - bucketOwnerFullControl
  - bucketOwnerRead
  - private
  - projectPrivate
  - publicRead

  如果不设置 object ACL 策略，则默认使用 private 策略。ACL 策略的详细介绍，参考 GCS 官方文档。

- `.spec.gcs.bucketAcl`：设置 bucket access-control list (ACL) 策略。

  GCS 支持以下几种 bucket ACL 策略：

  - `authenticatedRead`
  - `private`
  - `projectPrivate`
  - `publicRead`
  - `publicReadWrite`

  如果不设置 bucket ACL 策略，则默认策略为 `private`。ACL 策略的详细介绍，参考 GCS 官方文档。

7.4.2.1.5 Azure Blob Storage 存储字段介绍

- `.spec.azblob.secretName`：指定存储 Azure Blob Storage 用户账号认证信息的 Secret 名称。
- `.spec.azblob.container`：存储数据的 container 名字。
- `.spec.azblob.prefix`：如果设置了这个字段，则会使用这个字段来拼接在远端存储的存储路径 `azure://${.spec.azblob.container}/${.spec.azblob.prefix}/backupName`。
- `.spec.azblob.accessTier`：上传对象的存储类别。

  Azure Blob Storage 支持以下几种 `accessTier` 类型：

  - `Hot`
  - `Cool`
  - `Archive`

  如果不设置 `accessTier`，则默认使用 `Cool`。

7.4.2.1.6 Local 存储字段介绍

- `.spec.local.prefix`：持久卷存储目录。如果设置了这个字段，则会使用这个字段来拼接在持久卷的存储路径 `local://${.spec.local.volumeMount.mountPath}/${.spec.local.prefix}/`。
- `.spec.local.volume`：持久卷配置。
- `.spec.local.volumeMount`：持久卷挂载配置。

7.4.2.2 CompactBackup CR 字段介绍

对于 TiDB v9.0.0 及以上版本的集群，你可以使用 CompactBackup 加速日志恢复。要将日志备份数据压缩为结构化 SST 文件，你可以通过创建一个自定义的 CompactBackup CR 对象来描述一次备份任务。以下是 CompactBackup CR 各个字段的具体含义：

- .spec.startTs：指定日志压缩备份的起始时间戳。
- .spec.endTs：指定日志压缩备份的结束时间戳。
- .spec.concurrency：指定同时进行的压缩日志任务的最大数量，默认值为 4。
- .spec.maxRetryTimes：指定压缩任务失败的最大重试次数，默认值为 6。
- .spec.toolImage：指定 CompactBackup 使用的工具镜像。在 CompactBackup 中，唯一使用的工具镜像为 BR。使用 BR 备份时，你可以使用该字段指定 BR 的版本：
  - 如果未指定或者为空，默认使用镜像 pingcap/br:${tikv_version} 进行备份。
  - 如果指定了 BR 的版本，例如 .spec.toolImage: pingcap/br:v9.0.0，那么使用指定的版本镜像进行备份。
  - 如果指定了镜像但未指定版本，例如 .spec.toolImage: private/registry/br，那么使用镜像 private/registry/br:${tikv_version} 进行备份。
- .spec.env：指定运行压缩备份任务的 Pod 的环境变量信息。
- .spec.affinity：指定运行备份任务的 Pod 亲和性 (affinity) 配置。关于亲和性的详细说明，请参阅亲和性与反亲和性。
- .spec.tolerations：指定运行压缩备份任务的 Pod 能够被调度到带有与之匹配的污点 (Taint) 的节点上。关于污点与容忍度的更多说明，请参阅污点和容忍度。
- .spec.podSecurityContext：指定运行压缩备份任务的 Pod 的安全上下文配置，以支持非 root 用户运行 Pod。关于 podSecurityContext 的更多说明，请参阅以非 root 用户运行容器。
- .spec.priorityClassName：指定运行压缩备份任务的 Pod 的 priorityClass 的名称，用于设置运行优先级。关于 priorityClass 的更多说明，请参阅 Pod 优先级和抢占。
- .spec.imagePullSecrets：指定运行压缩备份任务的 Pod 使用的 imagePullSecrets。
- .spec.serviceAccount：指定恢复时使用的 ServiceAccount 名称。
- .spec.useKMS：指定恢复时是否使用 AWS-KMS 解密备份使用的 S3 存储密钥。
- .spec.br：BR 相关配置，详情请参阅BR 字段介绍。
- .spec.s3：S3 兼容存储相关配置，详情请参阅S3 字段介绍。
- .spec.gcs：GCS 存储相关配置，详情请参阅GCS 字段介绍。
- .spec.azblob：Azure Blob Storage 存储相关配置，详情请参阅Azure Blob Storage 字段介绍。

7.4.2.3 Restore CR 字段介绍

为了对 Kubernetes 上的 TiDB 集群进行数据恢复，用户可以通过创建一个自定义的 Restore CR 对象来描述一次恢复，具体恢复过程可参考备份与恢复简介中列出的文档。以下介绍 Restore CR 各个字段的具体含义。

- .spec.toolImage：用于指定 Restore 使用的工具镜像，例如，spec.toolImage: ↪ pingcap/br:v8.5.2。如果不指定，默认使用 pingcap/br:${tikv_version} 进行恢复。
- .spec.backupType：指定 Restore 类型，该字段仅在使用 BR 恢复时有效，目前支持以下三种类型，可以结合 .spec.tableFilter 配置表库过滤规则：
  - full：对 TiDB 集群所有的 database 数据执行备份。
  - db：对 TiDB 集群一个 database 的数据执行备份。
  - table：对 TiDB 集群表的数据执行备份。
- .spec.resources：指定运行恢复任务的 Pod 的资源请求与上限值。
- .spec.env：指定运行恢复任务的 Pod 的环境变量信息。
- .spec.affinity：指定运行恢复任务的 Pod 亲和性配置，关于 affinity 的使用说明，请参阅 Affinity & AntiAffinity。
- .spec.tolerations：指定运行恢复任务的 Pod 能够调度到带有与之匹配的污点 (Taint) 的节点上。关于污点与容忍度的更多说明，请参阅 Taints and Tolerations。
- .spec.podSecurityContext：指定运行恢复任务的 Pod 的安全上下文配置，允许 Pod 以非 root 用户的方式运行，关于 podSecurityContext 的更多说明，请参阅以非 root 用户运行容器。
- .spec.priorityClassName：指定运行恢复任务的 Pod 的 priorityClass 的名称，以设置运行优先级，关于 priorityClass 的更多说明，请参阅 Pod Priority and Preemption。
- .spec.imagePullSecrets：指定运行恢复任务的 Pod 的 imagePullSecrets
- .spec.serviceAccount：指定恢复时所使用的 ServiceAccount 名称。
- .spec.useKMS：指定恢复时是否使用 AWS-KMS 解密备份使用的 S3 存储密钥。
- .spec.storageClassName：指定恢复时所需的 PV 类型。
- .spec.storageSize：指定恢复集群时所需的 PV 大小。该值应大于 TiDB 集群备份的数据大小。
- .spec.tableFilter：恢复时指定让 BR 恢复符合 table-filter 规则 的表。默认情况下该字段可以不用配置。当不配置时，BR 会恢复备份文件中的所有数据库。

注意：

如果要使用排除规则 "!db.table" 导出除 db.table 的所有表，那么在 "!db.table" 前必须先添加 *.* 规则。如下面例子所示：

```
tableFilter:
- "*.*"
- "!db.table"
```

- .spec.br：BR 相关配置，具体介绍参考BR 字段介绍。
- .spec.s3：S3 兼容存储相关配置，具体介绍参考S3 字段介绍。
- .spec.gcs：GCS 存储相关配置，具体介绍参考GCS 字段介绍。
- .spec.azblob：Azure Blob Storage 存储相关配置，具体介绍参考Azure Blob Storage 字段介绍。
- .spec.local：持久卷存储相关配置，具体介绍参考Local 字段介绍。

7.4.2.4 BackupSchedule CR 字段介绍

backupSchedule 的配置由三部分组成。快照备份相关配置 backupTemplate，日志备份相关配置logBackupTemplate，backupSchedule 独有的配置。

- backupTemplate：快照备份相关配置。指定快照备份集群及远程存储相关的配置，字段和 Backup CR 中的 spec 一样，详细介绍可参考Backup CR 字段介绍。
- logBackupTemplate：日志备份相关配置。指定日志备份集群及远程存储相关的配置，字段和 Backup CR 中的 spec 一样，详细介绍可参考Backup CR 字段介绍，日志备份随 backupSchedule 创建、删除，且根据 .spec.maxReservedTime 进行回收。日志备份名称在 status.logBackup 中保存。
- compactBackupTemplate：压缩日志备份的配置模板，字段和 CompactBackup CR 中的 spec 一样，详细介绍可参考CompactBackup CR 字段介绍。压缩日志备份会随 backupSchedule 创建和删除，日志备份名称存储在 status.logBackup 中。压缩日志备份的存储设置应与同一 backupSchedule 中的 logBackupTemplate 保持一致。

> 注意：
> 若删除日志备份数据，需要先停止日志备份任务，避免由于未停止 TiKV 中的日志备份任务，造成资源浪费或者后续无法重新开启日志备份。

- backupSchedule 独有的配置：
  - .spec.maxBackups：一种备份保留策略，决定定时备份最多可保留的备份个数。超过该数目，就会将过时的备份删除。如果将该项设置为 0，则表示保留所有备份。
  - .spec.maxReservedTime：一种备份保留策略，按时间保留备份。例如将该参数设置为 24h，表示只保留最近 24 小时内的备份条目。超过这个时间的备份都会被清除。时间设置格式参考 func ParseDuration。如果同时设置 .spec.↪ maxBackups 和 .spec.maxReservedTime，则以 .spec.maxReservedTime 为准。
  - .spec.schedule：Cron 的时间调度格式。具体格式可参考 Cron。

- `.spec.pause`：是否暂停定时备份，默认为 `false`。如果将该值设置为 `true`，表示暂停定时备份，此时即使到了指定时间点，也不会进行备份。在定时备份暂停期间，备份 Garbage Collection (GC) 仍然正常进行。如需重新开启定时快照备份，将 `true` 改为`false`。由于目前日志备份暂不支持暂停，因此该配置对日志备份无效。

7.4.3 远程存储访问授权

本文详细描述了如何授权访问远程存储，以实现备份 TiDB 集群数据到远程存储或从远程存储恢复备份数据到 TiDB 集群。

7.4.3.1 AWS 账号授权

在 AWS 云环境中，不同类型的 Kubernetes 集群支持不同的权限授予方式。本文介绍以下三种授权方式：

- 通过 AccessKey 和 SecretKey 授权：适用于自建 Kubernetes 集群和 AWS EKS 集群。
- 通过 IAM 绑定 Pod 授权：适用于自建 Kubernetes 集群。
- 通过 IAM 绑定 ServiceAccount 授权：只适用于 AWS EKS 集群。

7.4.3.1.1 通过 AccessKey 和 SecretKey 授权

按照以下步骤配置 AccessKey 和 SecretKey，以授权访问兼容 S3 的存储服务：

1. 参考在 AWS 账户中创建 IAM 用户，创建一个 IAM 用户，并为其赋予需要的权限。由于备份与恢复操作需要访问 AWS 的 S3 存储，请赋予该 IAM 用户 `AmazonS3FullAccess` 权限。
2. 参考为自己创建访问密钥（控制台），为 IAM 用户创建访问密钥 (Access Key)。完成后，你将获得 AccessKey 和 SecretKey。
3. 使用以下命令创建一个名为 `s3-secret` 的 Kubernetes Secret，并填入上一步中获取的 AccessKey 和 SecretKey。该 Secret 用于存储访问 S3 兼容存储服务所需的凭证。

```
kubectl create secret generic s3-secret --from-literal=access_key=<your
    ↪ -access-key> --from-literal=secret_key=<your-secret-key> --
    ↪ namespace=<your-namespace>
```

4. AWS 客户端支持通过进程环境变量 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 获取与之相关联的用户权限。因此，可以通过为 Pod 设置相应的环境变量，使其具备访问兼容 S3 的存储服务的权限。

   以下示例展示如何通过Overlay 方式为 TiKVGroup 配置环境变量（TiFlashGroup 的配置方式相同）：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
  labels:
    pingcap.com/group: tikv
    pingcap.com/component: tikv
    pingcap.com/cluster: demo
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tikv
                env:
                  - name: "AWS_ACCESS_KEY_ID"
                    valueFrom:
                      secretKeyRef:
                        name: "s3-secret"
                        key: "access_key"
                  - name: "AWS_SECRET_ACCESS_KEY"
                    valueFrom:
                      secretKeyRef:
                        name: "s3-secret"
                        key: "secret_key"
```

以下是为 Backup 配置环境变量的示例：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup-s3
spec:
  env:
    - name: "AWS_ACCESS_KEY_ID"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
          key: "access_key"
    - name: "AWS_SECRET_ACCESS_KEY"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
```

```
        key: "secret_key"
```

以下是为 Restore 配置环境变量的示例：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: restore-s3
spec:
  env:
    - name: "AWS_ACCESS_KEY_ID"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
          key: "access_key"
    - name: "AWS_SECRET_ACCESS_KEY"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
          key: "secret_key"
```

7.4.3.1.2 通过 IAM 绑定 Pod 授权

通过 IAM 绑定 Pod 的授权方式由开源工具 kube2iam 提供，通过将 IAM 角色 绑定到 Pod，实现 Pod 中的进程继承 IAM 角色的访问权限。

> 注意：
>
> - kube2iam 仅适用于运行在 AWS EC2 实例上的 Kubernetes 集群，不支持其他类型的节点。
> - 使用该授权模式时，可以参考 kube2iam 文档在 Kubernetes 集群中创建 kube2iam 环境，并且部署 TiDB Operator 以及 TiDB 集群。
> - 该模式不适用于使用 `hostNetwork` 网络模式的 Pod。

通过 IAM 绑定 Pod 授权的步骤如下：

1. 参考 IAM 角色创建，为你的 AWS 账号创建一个 IAM 角色，并为其赋予 `AmazonS3FullAccess` 权限。
2. 通过Overlay 方式为目标组件（TiKV 或 TiFlash）绑定 IAM 角色。以下示例展示如何为 TiKVGroup 添加绑定：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          annotations:
            iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
```

注意：

你需要将 `arn:aws:iam::123456789012:role/user` 替换为步骤 1 中创建的 IAM 角色的实际 ARN。

7.4.3.1.3 通过 IAM 绑定 ServiceAccount 授权

通过将用户的 IAM 角色与 Kubernetes 中的 `ServiceAccount` 资源绑定，使用该 ServiceAccount 的 Pod 将继承该角色的权限。

通过 IAM 绑定 ServiceAccount 授权的步骤如下：

1. 参考 IAM 角色创建，为你的 AWS 账号创建一个 IAM 角色，并为其赋予 `AmazonS3FullAccess` 权限。
2. 参考为集群创建 IAM OIDC 提供商，为 EKS 集群创建一个 IAM OIDC 提供商。
3. 创建一个名为 `br-s3` 的 Kubernetes ServiceAccount，并参考为 Kubernetes 服务账户分配 IAM 角色将创建的 IAM 角色分配给该 ServiceAccount。
4. 通过Overlay 方式，将 ServiceAccount 绑定到 TiKVGroup 或 TiFlashGroup 的 Pod，以 TiKVGroup 为例：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
```

```
          serviceAccountName: br-s3
```

5. 在 Backup 或 Restore 中指定 serviceAccount，以 Backup 为例：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup-s3
spec:
  serviceAccount: br-s3
```

7.4.3.2 Google Cloud 账号授权

7.4.3.2.1 通过服务账号密钥授权

通过 Google Cloud 服务账号密钥授权的步骤如下：

1. 参考创建服务账号，创建一个服务账号，并生成服务账号密钥文件，保存为 google-credentials.json。

2. 使用以下命令创建一个名为 gcp-secret 的 Kubernetes Secret，用于存放 Google Cloud Storage 的访问凭证：

```
kubectl create secret generic gcp-secret --from-file=credentials=./
    ↪ google-credentials.json -n <your-namespace>
```

3. 参考将主账号添加到存储桶级层政策中，授予第 1 步中创建的服务账号对目标存储桶的访问权限，同时授予 roles/storage.objectUser 角色。

4. 为 Pod 设置环境变量，以 TiKVGroup 为例：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tikv
                env:
                  - name: "GOOGLE_APPLICATION_CREDENTIALS"
```

```
                valueFrom:
                  secretKeyRef:
                    name: "gcp-secret"
                    key: "credentials"
```

下面是给 Backup 配置使用该 Secret 的示例：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup-gcp
spec:
  gcs:
    secretName: gcp-secret
```

下面是给 Restore 配置使用该 Secret 的示例：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: restore-gcp
spec:
  gcs:
    secretName: gcp-secret
```

7.4.3.3 Azure 账号授权

在 Azure 云环境中，不同类型的 Kubernetes 集群可以通过不同方式授权访问 Azure Blob Storage。本文介绍以下两种常用的权限授予方式：

- 通过访问密钥授权：适用于所有类型的 Kubernetes 集群。
- 通过 Azure AD 授权：适用于需要更细粒度权限控制和密钥轮换的场景。

7.4.3.3.1 通过访问密钥授权

Azure 客户端支持从进程环境变量 `AZURE_STORAGE_ACCOUNT` 和 `AZURE_STORAGE_KEY` 中读取访问凭证。你可以通过以下步骤授权：

1. 使用以下命令创建一个名为 `azblob-secret` 的 Kubernetes Secret，将存储账户名称和密钥写入 Secret：

```
kubectl create secret generic azblob-secret \
  --from-literal=AZURE_STORAGE_ACCOUNT=<your-storage-account> \
  --from-literal=AZURE_STORAGE_KEY=<your-storage-key> \
  --namespace=<your-namespace>
```

2. 通过Overlay 方式，将该 Secret 绑定到 TiKVGroup 或 TiFlashGroup Pod 的环境变量，以 TiKVGroup 为例：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tikv
                env:
                  - name: "AZURE_STORAGE_ACCOUNT"
                    valueFrom:
                      secretKeyRef:
                        name: "azblob-secret"
                        key: "AZURE_STORAGE_ACCOUNT"
                  - name: "AZURE_STORAGE_KEY"
                    valueFrom:
                      secretKeyRef:
                        name: "azblob-secret"
                        key: "AZURE_STORAGE_KEY"
```

7.4.3.3.2 通过 Azure AD 授权

Azure 客户端支持通过环境变量 AZURE_STORAGE_ACCOUNT、AZURE_CLIENT_ID、AZURE_TENANT_ID 和 AZURE_CLIENT_SECRET 获取与之相关联的用户或角色的权限。此方式适合需要更高安全性及自动密钥轮换的场景。

1. 创建一个名为 azblob-secret-ad 的 Kubernetes Secret，存放用于访问 Azure Blob Storage 的凭证：

```
kubectl create secret generic azblob-secret-ad \
  --from-literal=AZURE_STORAGE_ACCOUNT=<your-storage-account> \
  --from-literal=AZURE_CLIENT_ID=<your-client-id> \
  --from-literal=AZURE_TENANT_ID=<your-tenant-id> \
  --from-literal=AZURE_CLIENT_SECRET=<your-client-secret> \
  --namespace=<your-namespace>
```

2. 通过Overlay 方式，将该 Secret 绑定到 TiKVGroup 或 TiFlashGroup Pod 的环境变量，以 TiKVGroup 为例：

注意：

- 通过 Azure AD 授权时，需确保服务主体已被授予目标存储账户的访问权限。
- 修改 Secret 后，需重启相关 Pod 以加载更新的环境变量。

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tikv
                envFrom:
                  secretRef:
                    name: "azblob-secret-ad"
```

7.4.4 使用 Amazon S3 兼容的存储

7.4.4.1 使用 BR 备份 TiDB 集群数据到兼容 S3 的存储

本文介绍如何将运行在 AWS Kubernetes 环境中的 TiDB 集群数据备份到 AWS 的存储上。其中包括以下两种备份方式：

- 快照备份。使用快照备份，你可以通过全量恢复将 TiDB 集群恢复到快照备份的时刻点。
- 日志备份。使用快照备份与日志备份，你可以通过快照备份与日志备份产生的备份数据将 TiDB 集群恢复到历史任意时刻点，即Point-in-Time Recovery (PITR)。

本文使用的备份方式基于 TiDB Operator 的 Custom Resource Definition (CRD) 实现，底层使用 BR 获取集群数据，然后再将数据上传到 AWS 的存储上。BR 全称为 Backup & Restore，是 TiDB 分布式备份恢复的命令行工具，用于对 TiDB 集群进行数据备份和恢复。

7.4.4.1.1 使用场景

如果你对数据备份有以下要求，可考虑使用 BR 的快照备份方式将 TiDB 集群数据以Ad-hoc 备份或定时快照备份的方式备份至兼容 S3 的存储上：

- 需要备份的数据量较大（大于 1 TiB），而且要求备份速度较快
- 需要直接备份数据的 SST 文件（键值对）

如果你对数据备份有以下要求，可考虑使用 BR 的日志备份方式将 TiDB 集群数据以Ad-hoc 备份的方式备份至兼容 S3 的存储上（同时也需要配合快照备份的数据，来更高效地恢复数据）：

- 需要在新集群上恢复备份集群的历史任意时刻点快照 (PITR)
- 数据的 RPO 在分钟级别

如有其他备份需求，请参考备份与恢复简介选择合适的备份方式。

注意：

- 快照备份只支持 TiDB v3.1 及以上版本。
- 日志备份只支持 TiDB v6.3 及以上版本。
- 使用 BR 备份出的数据只能恢复到 TiDB 数据库中，无法恢复到其他数据库中。

7.4.4.1.2 Ad-hoc 备份

Ad-hoc 备份支持快照备份，也支持启动和停止日志备份任务，以及清理日志备份数据等操作。

要进行 Ad-hoc 备份，你需要创建一个自定义的 `Backup` Custom Resource (CR) 对象来描述本次备份。创建好 `Backup` 对象后，TiDB Operator 根据这个对象自动完成具体的备份过程。如果备份过程中出现错误，程序不会自动重试，此时需要手动处理。

本文假设对部署在 Kubernetes `test1` 这个 namespace 中的 TiDB 集群 `demo1` 进行数据备份。下面是具体的操作过程。

前置条件：准备 Ad-hoc 备份环境

注意：

- BR 使用的 ServiceAccount 名称为固定值，必须为 `tidb-backup-manager`。
- 从 TiDB Operator v2 开始，`Backup` 和 `Restore` 等资源的 `apiGroup` 从 `pingcap.com` 修改为 `br.pingcap.com`。

1. 将以下内容保存为 backup-rbac.yaml 文件，用于创建所需的 RBAC 资源：

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. 执行以下命令在 namespace test1 中创建备份需要的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. 为 namespace test1 授予远程存储访问权限：

    - 如果使用 Amazon S3 来备份集群，可以使用三种方式授予权限，可参考文档AWS 账号授权。

- 如果使用其他兼容 S3 的存储来备份集群，例如 Ceph、MinIO，可以使用 AccessKey 和 SecretKey 授权的方式，可参考文档通过 AccessKey 和 SecretKey 授权。

快照备份

根据上一步选择的远程存储访问授权方式，你需要使用下面对应的方法将数据导出到兼容 S3 的存储上：

- 方法 1：如果通过了 accessKey 和 secretKey 的方式授权，你可以按照以下说明创建 Backup CR 备份集群数据：

```
kubectl apply -f full-backup-s3.yaml
```

full-backup-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: test1
spec:
  backupType: full
  br:
    cluster: demo1
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
    # options:
    # - --lastbackupts=420134118382108673
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

- 方法 2：如果通过了 IAM 绑定 Pod 的方式授权，你可以按照以下说明创建 Backup CR 备份集群数据：

```
kubectl apply -f full-backup-s3.yaml
```

full-backup-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  backupType: full
  br:
    cluster: demo1
    sendCredToTikv: false
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # options:
    # - --lastbackupts=420134118382108673
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

- 方法 3：如果通过了 IAM 绑定 ServiceAccount 的方式授权，你可以按照以下说明创建 Backup CR 备份集群数据：

```
kubectl apply -f full-backup-s3.yaml
```

full-backup-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
```

```
    cluster: demo1
    sendCredToTikv: false
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # options:
    # - --lastbackupts=420134118382108673
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

在配置 full-backup-s3.yaml 文件时，请参考以下信息：

- 自 TiDB Operator v1.1.6 版本起，如果需要增量备份，只需要在 spec.br.options 中指定上一次的备份时间戳 --lastbackupts 即可。有关增量备份的限制，可参考使用 BR 进行备份与恢复。
- Amazon S3 的 acl、endpoint、storageClass 配置项均可以省略。兼容 S3 的存储相关配置，请参考S3 存储字段介绍。
- .spec.br 中的一些参数是可选的，例如 logLevel、statusAddr 等。完整的 .spec.br 字段的详细解释，请参考BR 字段介绍。
- 如果你使用的 TiDB 为 v4.0.8 及以上版本，BR 会自动调整 tikv_gc_life_time 参数，不需要配置 spec.tikvGCLifeTime 和 spec.from 字段。
- 更多 Backup CR 字段的详细解释参考Backup CR 字段介绍。

查看快照备份的状态

创建好 Backup CR 后，TiDB Operator 会根据 Backup CR 自动开始备份。你可以通过如下命令查看备份状态：

```
kubectl get backup -n test1 -o wide
```

从上述命令的输出中，你可以找到描述名为 demo1-full-backup-s3 的 Backup CR 的如下信息，其中 COMMITTS 表示快照备份的时刻点：

```
NAME                 TYPE   MODE       STATUS    BACKUPPATH
    ↪                              COMMITTS           ...
demo1-full-backup-s3 full  snapshot  Complete  s3://my-bucket/my-full-backup-
    ↪ folder/ 436979621972148225 ...
```

日志备份

你可以使用一个 Backup CR 来描述日志备份任务的启动、停止以及清理日志备份数据等操作。日志备份对远程存储访问授权方式与快照备份一致。本节示例创建了名为 demo1-log-backup-s3 的 Backup CR，对远程存储访问授权方式仅以通过 accessKey 和 secretKey 的方式为例，具体操作如下所示。

logSubcommand 字段说明

在 Backup 自定义资源 (CR) 中，你可以使用 logSubcommand 字段控制日志备份任务的状态。logSubcommand 支持以下三个命令：

- log-start：该命令用于启动新的日志备份任务，或恢复已暂停的任务。使用此命令可以开始日志备份流程，或从暂停状态恢复任务。
- log-pause：该命令用于暂停当前正在进行的日志备份任务。暂停任务后，你可以使用 log-start 命令恢复任务。
- log-stop：该命令用于永久停止日志备份任务。执行此命令后，Backup CR 会进入停止状态，且无法再次启动。

这些命令提供了对日志备份任务生命周期的精细控制，支持启动、暂停、恢复和停止操作，帮助有效管理 Kubernetes 环境中的日志数据保留。

在 TiDB Operator v1.5.4、v1.6.0 及之前版本中，可以使用 logStop: true/false 字段来停止或启动日志备份任务。此字段在 TiDB Operator v2 中不再保留，推荐使用 logSubcommand 以确保配置清晰且一致。

启动日志备份

1. 在 test1 这个 namespace 中创建一个名为 demo1-log-backup-s3 的 Backup CR。

```
kubectl apply -f log-backup-s3.yaml
```

log-backup-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  br:
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
```

```
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
```

2. 等待启动操作完成：

```
kubectl get jobs -n test1
```

```
NAME                                  COMPLETIONS  ...
backup-demo1-log-backup-s3-log-start 1/1          ...
```

3. 查看新增的 Backup CR：

```
kubectl get backup -n test1
```

```
NAME                    MODE   STATUS   ....
demo1-log-backup-s3     log    Running  ....
```

查看日志备份的状态

通过查看 Backup CR 的信息，可查看日志备份的状态。

```
kubectl describe backup -n test1
```

从上述命令的输出中，你可以找到描述名为 demo1-log-backup-s3 的 Backup CR 的如下信息，其中 Log Checkpoint Ts 表示日志备份可恢复的最近时间点：

```
Status:
Backup Path: s3://my-bucket/my-log-backup-folder/
Commit Ts:  436568622965194754
Conditions:
    Last Transition Time: 2022-10-10T04:45:20Z
    Status:              True
    Type:                Scheduled
    Last Transition Time: 2022-10-10T04:45:31Z
    Status:              True
    Type:                Prepare
    Last Transition Time: 2022-10-10T04:45:31Z
    Status:              True
    Type:                Running
Log Checkpoint Ts:    436569119308644661
```

暂停日志备份

你可以通过将 Backup 自定义资源 (CR) 的 logSubcommand 字段设置为 log-pause 来暂停日志备份任务。下面以暂停启动日志备份中创建的名为 demo1-log-backup-s3 的 CR 为例。

```
kubectl edit backup demo1-log-backup-s3 -n test1
```

要暂停日志备份任务，只需将 logSubcommand 的值从 log-start 修改为 log-pause，然后保存并退出编辑器。修改后的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-s3 的 Backup CR 的 STATUS 从 Running 变成了 Pause：

```
kubectl get backup -n test1
```

```
NAME                    MODE    STATUS    ....
demo1-log-backup-s3     log     Pause     ....
```

恢复日志备份

如果日志备份任务已暂停，你可以通过将 logSubcommand 字段设置为 log-start 来恢复该任务。下面以恢复暂停日志备份中已暂停的 demo1-log-backup-s3 CR 为例。

> **Note:**
>
> 此操作仅适用于处于暂停状态 (Pause) 的任务，无法恢复状态为 Fail 或 Stopped 的任务。

```
kubectl edit backup demo1-log-backup-s3 -n test1
```

要恢复日志备份任务，只需将 logSubcommand 的值从 log-pause 更改为 log-start，然后保存并退出编辑器。修改后的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-s3 的 Backup CR 的 STATUS 从 Paused 状态变为 Running：

```
kubectl get backup -n test1
```

```
NAME                    MODE    STATUS    ....
demo1-log-backup-s3     log     Running   ....
```

停止日志备份

你可以通过将 Backup 自定义资源 (CR) 的 logSubcommand 字段设置为 log-stop 来停止日志备份。下面以停止启动日志备份中创建的名为 demo1-log-backup-s3 的 CR 为例。

```
kubectl edit backup demo1-log-backup-s3 -n test1
```

将 logSubcommand 的值修改为 log-stop，然后保存并退出编辑器。修改后的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
```

```
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-s3 的 Backup CR 的 STATUS 从 Running 变成了 Stopped：

```
kubectl get backup -n test1
```

```
NAME                    MODE    STATUS    ....
demo1-log-backup-s3     log     Stopped   ....
```

Stopped 是日志备份的终止状态。在此状态下，无法再次更改备份状态，但你仍然可以清理日志备份数据。

在 TiDB Operator v1.5.4、v1.6.0 及之前版本中，可以使用 logStop: true/false 字段来停止或启动日志备份任务。此字段仍然保留以确保向后兼容。

清理日志备份数据

1. 由于你在开启日志备份的时候已经创建了名为 demo1-log-backup-s3 的 Backup CR，因此可以直接更新该 Backup CR 的配置，来激活清理日志备份数据的操作。执行如下操作来清理 2022-10-10T15:21:00+08:00 之前的所有日志备份数据。

   ```
   kubectl edit backup demo1-log-backup-s3 -n test1
   ```

   在最后新增一行字段 spec.logTruncateUntil: "2022-10-10T15:21:00+08:00"，保存并退出。更新后的内容如下：

   ```
   ---
   apiVersion: br.pingcap.com/v1alpha1
   kind: Backup
   metadata:
     name: demo1-log-backup-s3
     namespace: test1
   spec:
     backupMode: log
     logSubcommand: log-start/log-pause/log-stop
     br:
   ```

```
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
  logTruncateUntil: "2022-10-10T15:21:00+08:00"
```

2. 等待清理操作完成：

```
kubectl get jobs -n test1
```

```
NAME                                    COMPLETIONS  ...
...
backup-demo1-log-backup-s3-log-truncate 1/1          ...
```

3. 查看 Backup CR 的信息：

```
kubectl describe backup -n test1
```

```
...
Log Success Truncate Until: 2022-10-10T15:21:00+08:00
...
```

也可以通过以下命令查看：

```
kubectl get backup -n test1 -o wide
```

```
NAME                 MODE      STATUS    ...  LOGTRUNCATEUNTIL
demo1-log-backup-s3 log       Stopped   ...  2022-10-10T15:21:00+08:00
```

压缩日志备份

对于 TiDB v9.0.0 及以上版本的集群，你可以使用 CompactBackup CR 将日志备份数据压缩为 SST 格式，以加速下游的日志恢复 (Point-in-time recovery, PITR)。

本节基于前文的日志备份示例，介绍如何使用压缩日志备份。

1. 在 test1 namespace 中创建一个名为 demo1-compact-backup 的 CompactBackup CR。

```
kubectl apply -f compact-backup-demo1.yaml
```

compact-backup-demo1.yaml 的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: CompactBackup
metadata:
  name: demo1-compact-backup
  namespace: test1
spec:
  startTs: "***"
  endTs: "***"
  concurrency: 8
  maxRetryTimes: 2
  br:
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
```

其中，startTs 和 endTs 指定 demo1-compact-backup 需要压缩的日志备份时间范围。任何包含至少一个该时间区间内写入的日志都会被送去压缩。因此，最终的压缩结果可能包含该时间范围之外的写入数据。

s3 设置应与需要压缩的日志备份的存储设置相同，CompactBackup 会读取相应地址的日志文件并进行压缩。

查看压缩日志备份状态

创建 CompactBackup CR 后，TiDB Operator 会自动开始压缩日志备份。你可以运行以下命令查看备份状态：

```
kubectl get cpbk -n test1
```

从上述命令的输出中，你可以找到描述名为 demo1-compact-backup 的 CompactBackup CR 的信息，输出示例如下：

```
NAME                    STATUS                PROGRESS
    ↪                                      MESSAGE
demo1-compact-backup Complete [READ_META(17/17),COMPACT_WORK(1291/1291)]
```

如果 STATUS 字段显示为 Complete 则代表压缩日志备份已经完成。

备份示例

备份全部集群数据

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

备份单个数据库的数据

以下示例中，备份 `db1` 数据库的数据。

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    sendCredToTikv: false
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

备份单张表的数据

以下示例中，备份 `db1.table1` 表的数据。

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    sendCredToTikv: false
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

使用表库过滤功能备份多张表的数据

以下示例中，备份 `db1.table1` 表和 `db1.table2` 表的数据。

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  # ...
  br:
    cluster: demo1
    sendCredToTikv: false
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

7.4.4.1.3 定时快照备份

你可以通过设置备份策略来对 TiDB 集群进行定时备份，同时设置备份的保留策略以避免产生过多的备份。定时快照备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次快照备份，定时快照备份底层通过 Ad-hoc 快照备份来实现。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

执行快照备份

依据准备 Ad-hoc 备份环境时所选择的远程存储访问授权方式，你需要使用下面对应的方法将数据定时备份到 Amazon S3 存储上：

- 方法 1：如果通过了 accessKey 和 secretKey 的方式授权，你可以按照以下说明创建 BackupSchedule CR，开启 TiDB 集群定时快照备份：

```
kubectl apply -f backup-scheduler-aws-s3.yaml
```

backup-scheduler-aws-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    # Clean outdated backup data based on maxBackups or maxReservedTime
       ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
      # sendCredToTikv: true
    s3:
      provider: aws
```

```
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

- 方法 2：如果通过了 IAM 绑定 Pod 的方式授权，你可以按照以下说明创建 BackupSchedule CR，开启 TiDB 集群定时快照备份：

```
kubectl apply -f backup-scheduler-aws-s3.yaml
```

backup-scheduler-aws-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: false
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

- 方法 3：如果通过了 IAM 绑定 ServiceAccount 的方式授权，你可以按照以下说明创建 BackupSchedule CR，开启 TiDB 集群定时快照备份：

```
kubectl apply -f backup-scheduler-aws-s3.yaml
```

backup-scheduler-aws-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    serviceAccount: tidb-backup-manager
    # Clean outdated backup data based on maxBackups or maxReservedTime
       ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: false
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

以上 backup-scheduler-aws-s3.yaml 文件配置示例中，backupSchedule 的配置由两部分组成。一部分是 backupSchedule 独有的配置，另一部分是 backupTemplate。

- 关于 backupSchedule 独有的配置项具体介绍，请参考BackupSchedule CR 字段介绍。

- backupTemplate 用于指定集群及远程存储相关的配置，字段和 Backup CR 中的 spec 一样，详细介绍可参考Backup CR 字段介绍。

定时快照备份创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

在进行集群恢复时，需要指定备份的路径，可以通过如下命令查看定时快照备份下面所有的备份条目，这些备份的名称以定时快照备份名称为前缀：

```
kubectl get bk -l tidb.pingcap.com/backup-schedule=demo1-backup-schedule-s3
    ↪ -n test1
```

7.4.4.1.4 集成管理定时快照备份和日志备份

BackupSchedule CR 可以集成管理 TiDB 集群的定时快照备份和日志备份。通过设置备份的保留时间，可以定期回收快照备份和日志备份，且能保证在保留期内可以通过快照备份和日志备份进行 PITR 恢复。

本节示例创建了名为 integrated-backup-schedule-s3 的 BackupSchedule CR，使用 accessKey 和 secretKey 的方式为例对远程存储进行访问授权，详细的授权方式参考AWS 账号授权。具体操作如下所示。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

创建 BackupSchedule

1. 在 test1 这个 namespace 中创建一个名为 integrated-backup-schedule-s3 的 BackupSchedule CR。

```
kubectl apply -f integrated-backup-schedule-s3.yaml
```

integrated-backup-schedule-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-s3
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
```

```
    br:
      cluster: demo1
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder-snapshot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder-log
```

以上 integrated-backup-schedule-s3.yaml 文件配置示例中，backupSchedule 的配置由三部分组成：backupSchedule 独有的配置，快照备份配置 backupTemplate，日志备份配置 logBackupTemplate。

关于 backupSchedule 配置项具体介绍，请参考BackupSchedule CR 字段介绍。

2. backupSchedule 创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

日志备份会随着 backupSchedule 创建，可以通过如下命令查看 backupSchedule 的 status.logBackup，即日志备份名称。

```
kubectl describe bks integrated-backup-schedule-s3 -n test1
```

3. 在进行集群恢复时，需要指定备份的路径。你可以通过如下命令查看定时快照备份下面所有的备份条目，在命令输出中 MODE 为 snapshot 的条目为快照备份，MODE 为 log 的条目为日志备份。

```
kubectl get bk -l tidb.pingcap.com/backup-schedule=integrated-backup-
    ↪ schedule-s3 -n test1
```

```
NAME                                                MODE       STATUS    ....
integrated-backup-schedule-s3-2023-03-08t02-45-00 snapshot Complete
    ↪ ....
log-integrated-backup-schedule-s3                   log        Running   ....
```

7.4.4.1.5 集成定时快照备份、日志备份和压缩日志备份

为了加快下游恢复速度，可以在 BackupSchedule CR 中添加压缩日志备份。压缩日志备份可以定期压缩远程存储中的日志备份文件。你必须先启用日志备份，才能使用压缩日志备份。本节基于上一节内容进行扩展。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

创建 BackupSchedule

1. 在 test1 这个 namespace 中创建一个名为 integrated-backup-schedule-s3 的 BackupSchedule CR。

```
kubectl apply -f integrated-backup-schedule-s3.yaml
```

integrated-backup-schedule-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-s3
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  compactInterval: "1h"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder-snapshot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    s3:
      provider: aws
```

```
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder-log
compactBackupTemplate:
  br:
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder-log
```

以上 integrated-backup-schedule-s3.yaml 文件配置示例中，backupSchedule 配置基于上一节内容，新增了 compactBackup 相关设置，主要改动如下：

- 新增 BackupSchedule.spec.compactInterval 字段，用于指定日志压缩备份的时间间隔。建议不要超过定时快照备份的间隔，并控制在定时快照备份间隔的二分之一至三分之一之间。
- 新增 BackupSchedule.spec.compactBackupTemplate 字段。请确保 BackupSchedule.spec.compactBackupTemplate.s3 配置与 BackupSchedule.spec.logBackupTemplate.s3 保持一致。

关于 backupSchedule 配置项具体介绍，请参考BackupSchedule CR 字段介绍。

2. backupSchedule 创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

压缩日志备份会随着 backupSchedule 创建，可以通过如下命令查看 CompactBackup CR 的信息。

```
kubectl get cpbk -n test1
```

7.4.4.1.6 删除备份的 Backup CR

如果你不再需要已备份的 Backup CR，请参考删除备份的 Backup CR。

7.4.4.1.7 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

7.4.4.2 使用 BR 恢复 S3 兼容存储上的备份数据

本文介绍如何将 S3 兼容存储上的备份数据恢复到 Kubernetes 环境中的 TiDB 集群，其中包括以下两种恢复方式：

- 全量恢复，可以将 TiDB 集群恢复到快照备份的时刻点。备份数据来自于快照备份。
- PITR 恢复，可以将 TiDB 集群恢复到历史任意时刻点。备份数据来自于快照备份和日志备份。

本文使用的恢复方式基于 TiDB Operator 的 Custom Resource Definition (CRD) 实现，底层使用 BR 进行数据恢复。BR 全称为 Backup & Restore，是 TiDB 分布式备份恢复的命令行工具，用于对 TiDB 集群进行数据备份和恢复。

PITR 全称为 Point-in-time recovery，该功能可以让你在新集群上恢复备份集群的历史任意时刻点的快照。使用 PITR 功能恢复时需要快照备份数据和日志备份数据。在恢复时，首先将快照备份的数据恢复到 TiDB 集群中，再以快照备份的时刻点作为起始时刻点，并指定任意恢复时刻点，将日志备份数据恢复到 TiDB 集群中。

> 注意：
>
> - BR 只支持 TiDB v3.1 及以上版本。
> - BR 的 PITR 恢复功能只支持 TiDB v6.3 及以上版本。
> - BR 恢复的数据无法被同步到下游，因为 BR 直接导入 SST/LOG 文件，而下游集群目前没有办法获得上游的 SST/LOG 文件。

7.4.4.2.1 全量恢复

本节示例将存储在 Amazon S3 上指定路径 `spec.s3.bucket` 存储桶中 `spec.s3.prefix` 文件夹下的快照备份数据恢复到 namespace `test1` 中的 TiDB 集群 `demo2`。以下是具体的操作过程。

前置条件：完成数据备份

本节假设 Amazon S3 中的桶 `my-bucket` 中文件夹 `my-full-backup-folder` 下存储着快照备份产生的备份数据。关于如何备份数据，请参考使用 BR 备份 TiDB 集群数据到兼容 S3 的存储。

第 1 步：准备恢复环境

使用 BR 将 S3 兼容存储上的备份数据恢复到 TiDB 前，请按照以下步骤准备恢复环境。

> 注意：

- BR 使用的 ServiceAccount 名称为固定值，必须为 `tidb-backup-manager`。
- 从 TiDB Operator v2 开始，`Backup`、`Restore` 等资源的 `apiGroup` 从 `pingcap.com` 修改为 `br.pingcap.com`。

1. 将以下内容保存为 `backup-rbac.yaml` 文件，用于创建所需的 RBAC 资源：

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. 执行以下命令在 namespace `test1` 中创建备份需要的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. 为 namespace `test1` 授予远程存储访问权限：

    - 如果使用 Amazon S3 来备份集群，可以使用三种方式授予权限，可参考文档AWS 账号授权。
    - 如果使用其他兼容 S3 的存储来备份集群，例如 Ceph、MinIO，可以使用 AccessKey 和 SecretKey 授权的方式，可参考文档通过 AccessKey 和 SecretKey 授权。

第 2 步：将指定备份数据恢复到 TiDB 集群

根据上一步选择的远程存储访问授权方式，你需要使用下面对应的方法将备份数据恢复到 TiDB：

- 方法 1：如果通过了 accessKey 和 secretKey 的方式授权，你可以按照以下说明创建 `Restore` CR 恢复集群数据：

```
kubectl apply -f restore-full-s3.yaml
```

`restore-full-s3.yaml` 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test1
spec:
  br:
    cluster: demo2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

- 方法 2：如果通过了 IAM 绑定 Pod 的方式授权，你可以按照以下说明创建 Restore CR 恢复集群数据：

```
kubectl apply -f restore-full-s3.yaml
```

restore-full-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  br:
    cluster: demo2
    sendCredToTikv: false
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

- 方法 3：如果通过了 IAM 绑定 ServiceAccount 的方式授权，你可以按照以下说明创建 Restore CR 恢复集群数据：

```
kubectl apply -f restore-full-s3.yaml
```

restore-full-s3.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test1
spec:
  serviceAccount: tidb-backup-manager
  br:
```

```
    cluster: demo2
    sendCredToTikv: false
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

在配置 restore-full-s3.yaml 文件时，请参考以下信息：

- 关于兼容 S3 的存储相关配置，请参考S3 存储字段介绍。
- .spec.br 中的一些参数为可选项，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释，请参考BR 字段介绍。
- 如果你使用的 TiDB 为 v4.0.8 及以上版本，BR 会自动调整 tikv_gc_life_time 参数，不需要在 Restore CR 中配置 spec.to 字段。
- 更多 Restore CR 字段的详细解释，请参考Restore CR 字段介绍。

创建好 Restore CR 后，可通过以下命令查看恢复的状态：

```
kubectl get restore -n restore-test -o wide
```

```
NAME              STATUS    ...
demo2-restore-s3  Complete  ...
```

7.4.4.2.2 PITR 恢复

本节示例在 namespace test1 中的 TiDB 集群 demo3 上执行 PITR 恢复，分为以下两步：

1. 使用 spec.pitrFullBackupStorageProvider.s3.bucket 存储桶中 spec.pitrFullBackupStorageProvider.s3.prefix 文件夹下的快照备份数据，将集群恢复到快照备份的时刻点。
2. 使用 spec.s3.bucket 存储桶中 spec.s3.prefix 文件夹下的日志备份的增量数据，将集群恢复到备份集群的历史任意时刻点。

下面是具体的操作过程。

前置条件：完成数据备份

本节假设 Amazon S3 中的桶 my-bucket 中存在两份备份数据，分别是：

- 在日志备份期间，进行快照备份产生的备份数据，存储在 `my-full-backup-folder-pitr` 文件夹下。
- 日志备份产生的备份数据，存储在 `my-log-backup-folder-pitr` 文件夹下。

关于如何备份数据，请参考使用 BR 备份 TiDB 集群数据到兼容 S3 的存储。

> **注意：**
>
> 指定的恢复时间点需要在快照备份时刻点之后，日志备份 `checkpoint-ts` 之前。

第 1 步：准备恢复环境

参考使用 BR 恢复 S3 兼容存储上的备份数据。

第 2 步：将指定备份数据恢复到 TiDB 集群

本节示例中首先将快照备份恢复到集群中，因此 PITR 的恢复时刻点需要在快照备份的时刻点之后，并在日志备份的最新恢复点之前。PITR 恢复对远程存储访问授权方式与快照备份恢复一致。本节示例对远程存储访问授权方式仅以通过 accessKey 和 secretKey 的方式为例，具体步骤如下：

1. 在 `restore-test` 这个 namespace 中产生一个名为 `demo3-restore-s3` 的 `Restore` CR，并指定恢复到 `2022-10-10T17:21:00+08:00`：

```
kubectl apply -f restore-point-s3.yaml
```

`restore-point-s3.yaml` 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-s3
  namespace: test1
spec:
  restoreMode: pitr
  br:
    cluster: demo3
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder-pitr
```

```
  pitrRestoredTs: "2022-10-10T17:21:00+08:00"
  pitrFullBackupStorageProvider:
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-full-backup-folder-pitr
```

在配置 restore-point-s3.yaml 文件时，请参考以下信息：

- spec.restoreMode：在进行 PITR 恢复时，需要设置值为 pitr。默认值为 snapshot，即进行全量恢复。

2. 查看恢复的状态，等待恢复操作完成：

```
kubectl get jobs -n restore-test
```

```
NAME                       COMPLETIONS  ...
restore-demo3-restore-s3 1/1            ...
```

也可通过以下命令查看恢复的状态：

```
kubectl get restore -n restore-test -o wide
```

```
NAME             STATUS    ...
demo3-restore-s3 Complete  ...
```

7.4.4.2.3 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

7.4.5 使用 Google Cloud Storage

7.4.5.1 使用 BR 备份 TiDB 集群到 GCS

本文介绍如何将运行在 Kubernetes 环境中的 TiDB 集群数据备份到 Google Cloud Storage (GCS) 的存储上。其中包括以下两种备份方式：

- 快照备份。使用快照备份，你可以通过全量恢复将 TiDB 集群恢复到快照备份的时刻点。
- 日志备份。使用快照备份与日志备份，你可以通过快照备份与日志备份产生的备份数据将 TiDB 集群恢复到历史任意时刻点，即Point-in-Time Recovery (PITR)。

本文使用的备份方式基于 TiDB Operator 的 Custom Resource Definition (CRD) 实现，底层使用 BR 获取集群数据，然后再将数据上传到远端 GCS。BR 全称为 Backup & Restore，是 TiDB 分布式备份恢复的命令行工具，用于对 TiDB 集群进行数据备份和恢复。

7.4.5.1.1 使用场景

如果你对数据备份有以下要求，可考虑使用 BR 的快照备份方式将 TiDB 集群数据以Ad-hoc 备份或定时快照备份的方式备份至 GCS 上：

- 需要备份的数据量较大（大于 1 TiB），而且要求备份速度较快
- 需要直接备份数据的 SST 文件（键值对）

如果你对数据备份有以下要求，可考虑使用 BR 的日志备份方式将 TiDB 集群数据以Ad-hoc 备份的方式备份至 GCS 上（同时也需要配合快照备份的数据，来更高效地恢复数据）：

- 需要在新集群上恢复备份集群的历史任意时刻点快照 (PITR)
- 数据的 RPO 在分钟级别

如有其他备份需求，请参考备份与恢复简介选择合适的备份方式。

> 注意：
>
> - 快照备份只支持 TiDB v3.1 及以上版本。
> - 日志备份只支持 TiDB v6.3 及以上版本。
> - 使用 BR 备份出的数据只能恢复到 TiDB 数据库中，无法恢复到其他数据库中。

7.4.5.1.2 Ad-hoc 备份

Ad-hoc 备份支持快照备份，也支持启动和停止日志备份任务，以及清理日志备份数据等操作。

要进行 Ad-hoc 备份，你需要创建一个自定义的 `Backup` custom resource (CR) 对象来描述本次备份。创建好 `Backup` 对象后，TiDB Operator 根据这个对象自动完成具体的备份过程。如果备份过程中出现错误，程序不会自动重试，此时需要手动处理。

本文档假设对部署在 Kubernetes `test1` 这个 namespace 中的 TiDB 集群 `demo1` 进行数据备份。下面是具体的操作过程。

前置条件：准备 Ad-hoc 备份环境

> 注意：

- BR 使用的 ServiceAccount 名称为固定值，必须为 `tidb-backup-manager`。
- 从 TiDB Operator v2 开始，`Backup` 和 `Restore` 等资源的 `apiGroup` 从 `pingcap.com` 修改为 `br.pingcap.com`。

1. 将以下内容保存为 `backup-rbac.yaml` 文件，用于创建所需的 RBAC 资源：

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. 执行以下命令在 namespace test1 中创建备份需要的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. 为 namespace test1 授予远程存储访问权限：

   参考GCS 账号授权，授权访问 GCS 远程存储。

快照备份

创建 Backup CR，将数据备份到 GCS：

```
kubectl apply -f full-backup-gcs.yaml
```

full-backup-gcs.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-gcs
  namespace: test1
spec:
  # backupType: full
  br:
    cluster: demo1
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
    # sendCredToTikv: true
    # options:
    # - --lastbackupts=420134118382108673
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-full-backup-folder
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

在配置 full-backup-gcs.yaml 文件时，请参考以下信息：

- 自 TiDB Operator v1.1.6 版本起，如果需要增量备份，只需要在 spec.br.options 中指定上一次的备份时间戳 --lastbackupts 即可。有关增量备份的限制，可参考使用 BR 进行备份与恢复。

- .spec.br 中的一些参数是可选的，例如 logLevel、statusAddr 等。完整的 .spec.br 字段的详细解释，请参考BR 字段介绍。
- spec.gcs 中的一些参数为可选项，如 location、objectAcl、storageClass。GCS 存储相关配置参考GCS 存储字段介绍。
- 如果你使用的 TiDB 为 v4.0.8 及以上版本，BR 会自动调整 tikv_gc_life_time 参数，不需要配置 spec.tikvGCLifeTime 和 spec.from 字段。
- 更多 Backup CR 字段的详细解释，请参考Backup CR 字段介绍。

查看快照备份的状态

创建好 Backup CR 后，TiDB Operator 会根据 Backup CR 自动开始备份。你可以通过如下命令查看备份状态：

```
kubectl get backup -n test1 -o wide
```

从上述命令的输出中，你可以找到描述名为 demo1-full-backup-gcs 的 Backup CR 的如下信息，其中 COMMITTS 表示快照备份的时刻点：

```
NAME                  TYPE   MODE      STATUS    BACKUPPATH
    ↪                              COMMITTS           ...
demo1-full-backup-gcs full snapshot  Complete  gcs://my-bucket/my-full-
    ↪ backup-folder/ 436979621972148225 ...
```

日志备份

你可以使用一个 Backup CR 来描述日志备份任务的启动、停止以及清理日志备份数据等操作。日志备份对远程存储访问授权方式与快照备份一致。本节示例创建了名为 demo1-log-backup-gcs 的 Backup CR，具体操作如下所示。

logSubcommand 字段说明

在 Backup 自定义资源 (CR) 中，你可以使用 logSubcommand 字段控制日志备份任务的状态。logSubcommand 支持以下三个命令：

- log-start：该命令用于启动新的日志备份任务，或恢复已暂停的任务。使用此命令可以开始日志备份流程，或从暂停状态恢复任务。
- log-pause：该命令用于暂停当前正在进行的日志备份任务。暂停任务后，你可以使用 log-start 命令恢复任务。
- log-stop：该命令用于永久停止日志备份任务。执行此命令后，Backup CR 会进入停止状态，且无法再次启动。

这些命令提供了对日志备份任务生命周期的精细控制，支持启动、暂停、恢复和停止操作，帮助有效管理 Kubernetes 环境中的日志数据保留。

在 TiDB Operator v1.5.4、v1.6.0 及之前版本中，可以使用 logStop: true/false 字段来停止或启动日志备份任务。此字段在 TiDB Operator v2 中不再保留，推荐使用 logSubcommand 以确保配置清晰且一致。

启动日志备份

1. 在 test1 这个 namespace 中创建一个名为 demo1-log-backup-gcs 的 Backup CR。

```
kubectl apply -f log-backup-gcs.yaml
```

log-backup-gcs.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

2. 等待启动操作完成：

```
kubectl get jobs -n test1
```

```
NAME                                  COMPLETIONS  ...
backup-demo1-log-backup-gcs-log-start 1/1          ...
```

3. 查看新增的 Backup CR：

```
kubectl get backup -n test1
```

```
NAME                     MODE   STATUS   ....
demo1-log-backup-gcs     log    Running  ....
```

查看日志备份的状态

通过查看 Backup CR 的信息，可查看日志备份的状态。

```
kubectl describe backup -n test1
```

从上述命令的输出中，你可以找到描述名为 demo1-log-backup-gcs 的 Backup CR 的如下信息，其中 Log Checkpoint Ts 表示日志备份可恢复的最近时间点：

```
Status:
Backup Path: gcs://my-bucket/my-log-backup-folder/
Commit Ts:  436568622965194754
Conditions:
    Last Transition Time: 2022-10-10T04:45:20Z
    Status:              True
    Type:                Scheduled
    Last Transition Time: 2022-10-10T04:45:31Z
    Status:              True
    Type:                Prepare
    Last Transition Time: 2022-10-10T04:45:31Z
    Status:              True
    Type:                Running
Log Checkpoint Ts:    436569119308644661
```

暂停日志备份

你可以通过将 Backup 自定义资源 (CR) 的 logSubcommand 字段设置为 log-pause 来暂停日志备份任务。下面以暂停启动日志备份中创建的名为 demo1-log-backup-gcs 的 CR 为例。

```
kubectl edit backup demo1-log-backup-gcs -n test1
```

要暂停日志备份任务，只需将 logSubcommand 的值从 log-start 修改为 log-pause，然后保存并退出编辑器。修改后的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-gcs 的 Backup CR 的 STATUS 从 Running 变成了 Pause：

```
kubectl get backup -n test1
```

```
NAME                      MODE     STATUS    ....
demo1-log-backup-gcs       log      Pause     ....
```

恢复日志备份

如果日志备份任务已暂停，你可以通过将 logSubcommand 字段设置为 log-start 来恢复该任务。下面以恢复暂停日志备份中已暂停的 demo1-log-backup-gcs CR 为例。

> **Note:**
>
> 此操作仅适用于处于暂停状态 (Pause) 的任务，无法恢复状态为 Fail 或 Stopped 的任务。

```
kubectl edit backup demo1-log-backup-gcs -n test1
```

要恢复日志备份任务，只需将 logSubcommand 的值从 log-pause 更改为 log-start，然后保存并退出编辑器。修改后的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-gcs 的 Backup CR 的 STATUS 从 Paused 状态变为 Running：

```
kubectl get backup -n test1
```

```
NAME                    MODE    STATUS   ....
demo1-log-backup-gcs     log     Running  ....
```

停止日志备份

你可以通过将 Backup 自定义资源 (CR) 的 logSubcommand 字段设置为 log-stop 来停止日志备份。下面以停止启动日志备份中创建的名为 demo1-log-backup-gcs 的 CR 为例。

```
kubectl edit backup demo1-log-backup-gcs -n test1
```

将 logSubcommand 的值修改为 log-stop，然后保存并退出编辑器。修改后的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-gcs 的 Backup CR 的 STATUS 从 Running 变成了 Stopped：

```
kubectl get backup -n test1
```

```
NAME                    MODE    STATUS   ....
demo1-log-backup-gcs    log     Stopped  ....
```

Stopped 是日志备份的终止状态。在此状态下，无法再次更改备份状态，但你仍然可以清理日志备份数据。

在 TiDB Operator v1.5.4、v1.6.0 及之前版本中，可以使用 logStop: true/false 字段来停止或启动日志备份任务。此字段仍然保留以确保向后兼容。

清理日志备份数据

1. 由于你在开启日志备份的时候已经创建了名为 demo1-log-backup-gcs 的 Backup CR，因此可以直接更新该 Backup CR 的配置，来激活清理日志备份数据的操作。执行如下操作来清理 2022-10-10T15:21:00+08:00 之前的所有日志备份数据。

```
kubectl edit backup demo1-log-backup-gcs -n test1
```

在最后新增一行字段 spec.logTruncateUntil: "2022-10-10T15:21:00+08:00"，保存并退出。更新后的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start/log-pause/log-stop
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
  logTruncateUntil: "2022-10-10T15:21:00+08:00"
```

2. 等待清理操作完成：

```
kubectl get jobs -n test1
```

```
NAME                                        COMPLETIONS  ...
...
backup-demo1-log-backup-gcs-log-truncate 1/1      ...
```

3. 查看 Backup CR 的信息：

```
kubectl describe backup -n test1
```

```
...
Log Success Truncate Until: 2022-10-10T15:21:00+08:00
...
```

也可以通过以下命令查看：

```
kubectl get backup -n test1 -o wide
```

```
NAME                  MODE      STATUS    ...   LOGTRUNCATEUNTIL
demo1-log-backup-gcs  log       Stopped   ...   2022-10-10T15:21:00+08:00
```

压缩日志备份

对于 TiDB v9.0.0 及以上版本的集群，你可以使用 CompactBackup CR 将日志备份数据压缩为 SST 格式，以加速下游的日志恢复 (Point-in-time recovery, PITR)。

本节基于前文的日志备份示例，介绍如何使用压缩日志备份。

在 test1 namespace 中创建一个名为 demo1-compact-backup 的 CompactBackup CR。

```
kubectl apply -f compact-backup-demo1.yaml
```

compact-backup-demo1.yaml 的内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: CompactBackup
metadata:
  name: demo1-compact-backup
  namespace: test1
spec:
  startTs: "***"
  endTs: "***"
  concurrency: 8
  maxRetryTimes: 2
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

其中，startTs 和 endTs 指定 demo1-compact-backup 需要压缩的日志备份时间范围。任何包含至少一个该时间区间内写入的日志都会被送去压缩。因此，最终的压缩结果可能包含该时间范围之外的写入数据。

gcs 设置应与需要压缩的日志备份的存储设置相同，CompactBackup 会读取相应地址的日志文件并进行压缩。

查看压缩日志备份状态

创建 CompactBackup CR 后，TiDB Operator 会自动开始压缩日志备份。你可以运行以下命令查看备份状态：

```
kubectl get cpbk -n test1
```

从上述命令的输出中，你可以找到描述名为 demo1-compact-backup 的 CompactBackup CR 的信息，输出示例如下：

```
NAME                    STATUS                  PROGRESS
    ↪                                       MESSAGE
demo1-compact-backup Complete [READ_META(17/17),COMPACT_WORK(1291/1291)]
```

如果 STATUS 字段显示为 Complete 则代表压缩日志备份已经完成。

备份示例

备份全部集群数据

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  br:
    cluster: demo1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

备份单个数据库的数据

以下示例中，备份 db1 数据库的数据。

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  tableFilter:
```

```
  - "db1.*"
  br:
    cluster: demo1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

备份单张表的数据

以下示例中，备份 `db1.table1` 表的数据。

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

使用表库过滤功能备份多张表的数据

以下示例中，备份 `db1.table1` 表和 `db1.table2` 表的数据。

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
```

```
spec:
  # backupType: full
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  br:
    cluster: demo1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

7.4.5.1.3 定时快照备份

你可以通过设置备份策略来对 TiDB 集群进行定时备份，同时设置备份的保留策略以避免产生过多的备份。定时快照备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次快照备份，定时快照备份底层通过 Ad-hoc 快照备份来实现。下面是创建定时快照备份的具体步骤。

前置条件：准备定时快照备份环境

同Ad-hoc 快照备份环境准备。

执行快照备份

1. 创建 BackupSchedule CR，开启 TiDB 集群的定时快照备份，将数据备份到 GCS：

```
kubectl apply -f backup-schedule-gcs.yaml
```

backup-schedule-gcs.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-gcs
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
```

```
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      # logLevel: info
      # statusAddr: ${status-addr}
      # concurrency: 4
      # rateLimit: 0
      # checksum: true
      # sendCredToTikv: true
    gcs:
      secretName: gcs-secret
      projectId: ${project_id}
      bucket: ${bucket}
      prefix: ${prefix}
      # location: us-east1
      # storageClass: STANDARD_IA
      # objectAcl: private
```

从以上 backup-schedule-gcs.yaml 文件配置示例可知，backupSchedule 的配置由两部分组成。一部分是 backupSchedule 独有的配置，另一部分是 backupTemplate。

- 关于 backupSchedule 独有的配置项具体介绍，请参考BackupSchedule CR 字段介绍。
- backupTemplate 用于指定集群及远程存储相关的配置，字段和 Backup CR 中的 spec 一样，详细介绍可参考Backup CR 字段介绍。

2. 定时快照备份创建完成后，通过以下命令查看备份的状态：

```
kubectl get bks -n test1 -o wide
```

查看定时快照备份下面所有的备份条目：

```
kubectl get bk -l tidb.pingcap.com/backup-schedule=demo1-backup-
  ↪ schedule-gcs -n test1
```

7.4.5.1.4 集成管理定时快照备份和日志备份

BackupSchedule CR 可以集成管理 TiDB 集群的定时快照备份和日志备份。通过设置备份的保留时间，可以定期回收快照备份和日志备份，且能保证在保留期内可以通过快照备份和日志备份进行 PITR 恢复。

本节示例创建了名为 integrated-backup-schedule-gcs 的 BackupSchedule CR，其中访问 GCS 远程存储的方式参考GCS 账号授权，具体操作如下所示。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

创建 BackupSchedule

1. 在 test1 这个 namespace 中创建一个名为 integrated-backup-schedule-gcs 的 BackupSchedule CR。

```
kubectl apply -f integrated-backup-scheduler-gcs.yaml
```

integrated-backup-scheduler-gcs.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-gcs
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-snapshot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-log
```

以上 integrated-backup-scheduler-gcs.yaml 文件配置示例中，backupSchedule 的配置由三部分组成：backupSchedule 独有的配置，快照备份配置 backupTemplate，日志备份配置 logBackupTemplate。

关于 backupSchedule 配置项具体介绍，请参考BackupSchedule CR 字段介绍。

2. backupSchedule 创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

日志备份会随着 backupSchedule 创建，可以通过如下命令查看 backupSchedule 的 status.logBackup，即日志备份名称。

```
kubectl describe bks integrated-backup-schedule-gcs -n test1
```

3. 在进行集群恢复时，需要指定备份的路径。你可以通过如下命令查看定时快照备份下面所有的备份条目，在命令输出中 MODE 为 snapshot 的条目为快照备份，MODE 为 log 的条目为日志备份。

```
kubectl get bk -l tidb.pingcap.com/backup-schedule=integrated-backup-
    ↪ schedule-gcs -n test1
```

```
NAME                                                    MODE       STATUS
    ↪ ....
integrated-backup-schedule-gcs-2023-03-08t02-50-00  snapshot  Complete
    ↪ ....
log-integrated-backup-schedule-gcs                      log        Running
    ↪ ....
```

7.4.5.1.5 集成定时快照备份、日志备份和压缩日志备份

为了加快下游恢复速度，可以在 BackupSchedule CR 中添加压缩日志备份。压缩日志备份可以定期压缩远程存储中的日志备份文件。你必须先启用日志备份，才能使用压缩日志备份。本节基于上一节内容进行扩展。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

创建 BackupSchedule

1. 在 test1 这个 namespace 中创建一个名为 integrated-backup-schedule-gcs 的 BackupSchedule CR。

```
kubectl apply -f integrated-backup-scheduler-gcs.yaml
```

integrated-backup-scheduler-gcs.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-gcs
  namespace: test1
```

```
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  compactInterval: "1h"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-snapshot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-log
  compactBackupTemplate:
    br:
      cluster: demo1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-log
```

以上 integrated-backup-schedule-gcs.yaml 文件配置示例中，backupSchedule 配置基于上一节内容，新增了 compactBackup 相关设置，主要改动如下：

- 新增 BackupSchedule.spec.compactInterval 字段，用于指定日志压缩备份的时间间隔。建议不要超过定时快照备份的间隔，并控制在定时快照备份间隔的二分之一至三分之一之间。
- 新增 BackupSchedule.spec.compactBackupTemplate 字段。请确保 BackupSchedule.spec.compactBackupTemplate.gcs 配置与 BackupSchedule.spec.logBackupTemplate.gcs 保持一致。

关于 backupSchedule 配置项具体介绍，请参考BackupSchedule CR 字段介绍。

2. backupSchedule 创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

压缩日志备份会随着 backupSchedule 创建，可以通过如下命令查看 CompactBackup CR 的信息。

```
kubectl get cpbk -n test1
```

7.4.5.1.6 删除备份的 Backup CR

如果你不再需要已备份的 Backup CR，请参考删除备份的 Backup CR。

7.4.5.1.7 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

7.4.5.2 使用 BR 恢复 GCS 上的备份数据

本文介绍如何将存储在 Google Cloud Storage (GCS) 上的备份数据恢复到 Kubernetes 环境中的 TiDB 集群，其中包括以下两种恢复方式：

- 全量恢复，可以将 TiDB 集群恢复到快照备份的时刻点。备份数据来自于快照备份。
- PITR 恢复，可以将 TiDB 集群恢复到历史任意时刻点。备份数据来自于快照备份和日志备份。

本文使用的恢复方式基于 TiDB Operator 的 Custom Resource Definition (CRD) 实现，底层使用 BR 进行数据恢复。BR 全称为 Backup & Restore，是 TiDB 分布式备份恢复的命令行工具，用于对 TiDB 集群进行数据备份和恢复。

PITR 全称为 Point-in-time recovery，该功能可以让你在新集群上恢复备份集群的历史任意时刻点的快照。使用 PITR 功能恢复时需要快照备份数据和日志备份数据。在恢复时，首先将快照备份的数据恢复到 TiDB 集群中，再以快照备份的时刻点作为起始时刻点，并指定任意恢复时刻点，将日志备份数据恢复到 TiDB 集群中。

注意：

- BR 只支持 TiDB v3.1 及以上版本。
- BR 的 PITR 恢复功能只支持 TiDB v6.3 及以上版本。
- BR 恢复的数据无法被同步到下游，因为 BR 直接导入 SST/LOG 文件，而下游集群目前没有办法获得上游的 SST/LOG 文件。

7.4.5.2.1 全量恢复

本节示例将存储在 GCS 上指定路径 spec.gcs.bucket 存储桶中 spec.gcs.prefix 文件夹下的快照备份数据恢复到 namespace test1 中的 TiDB 集群 demo2。以下是具体的操作过程。

前置条件：完成数据备份

本节假设 GCS 中的桶 my-bucket 中文件夹 my-full-backup-folder 下存储着快照备份产生的备份数据。关于如何备份数据，请参考使用 BR 备份 TiDB 集群到 GCS。

第 1 步：准备恢复环境

使用 BR 将 GCS 上的备份数据恢复到 TiDB 前，你需要准备恢复环境，并拥有数据库的相关权限。

> 注意：
>
> - BR 使用的 ServiceAccount 名称为固定值，必须为 tidb-backup-manager。
> - 从 TiDB Operator v2 开始，Backup、Restore 等资源的 apiGroup 从 pingcap.com 修改为 br.pingcap.com。

1. 将以下内容保存为 backup-rbac.yaml 文件，用于创建所需的 RBAC 资源：

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager
```

```
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. 执行以下命令在 namespace test1 中创建备份需要的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. 为 namespace test1 授予远程存储访问权限：

   参考GCS 账号授权，授权访问 GCS 远程存储。

第 2 步：将指定备份数据恢复到 TiDB 集群

1. 创建 Restore Custom Resource (CR)，将指定的备份数据恢复至 TiDB 集群：

```
kubectl apply -f restore-full-gcs.yaml
```

restore-full-gcs.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-gcs
  namespace: test1
spec:
  # backupType: full
  br:
    cluster: demo2
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
```

```
    # checksum: true
    # sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-full-backup-folder
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

在配置 restore-full-gcs.yaml 文件时，请参考以下信息：

- 关于 GCS 存储相关配置，请参考GCS 存储字段介绍。
- .spec.br 中的一些参数为可选项，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释，请参考BR 字段介绍。
- 如果你使用的 TiDB 为 v4.0.8 及以上版本，BR 会自动调整 tikv_gc_life_time 参数，不需要在 Restore CR 中配置 spec.to 字段。
- 更多 Restore CR 字段的详细解释，请参考Restore CR 字段介绍。

2. 创建好 Restore CR 后，通过以下命令查看恢复的状态：

```
kubectl get restore -n test1 -o wide
```

```
NAME               STATUS    ...
demo2-restore-gcs Complete ...
```

7.4.5.2.2 PITR 恢复

本节示例在 namespace test1 中的 TiDB 集群 demo3 上执行 PITR 恢复，分为以下两步：

1. 使用 spec.pitrFullBackupStorageProvider.gcs.bucket 存储桶中 spec.pitrFullBackupStorageProvider.gcs.prefix 文件夹下的快照备份数据，将集群恢复到快照备份的时刻点。
2. 使用 spec.gcs.bucket 存储桶中 spec.gcs.prefix 文件夹下的日志备份的增量数据，将集群恢复到备份集群的历史任意时刻点。

下面是具体的操作过程。

前置条件：完成数据备份

本节假设 GCS 中的桶 my-bucket 中存在两份备份数据，分别是：

- 在日志备份期间，进行快照备份产生的备份数据，存储在 `my-full-backup-folder-pitr` 文件夹下。
- 日志备份产生的备份数据，存储在 `my-log-backup-folder-pitr` 文件夹下。

关于如何备份数据，请参考使用 BR 备份 TiDB 集群到 GCS。

> **注意：**
>
> 指定的恢复时间点需要在快照备份时刻点之后，日志备份 `checkpoint-ts` 之前。

第 1 步：准备恢复环境

参考使用 BR 恢复 GCS 上的备份数据。

第 2 步：将指定备份数据恢复到 TiDB 集群

本节示例中首先将快照备份恢复到集群中，因此 PITR 的恢复时刻点需要在快照备份的时刻点之后，并在日志备份的最新恢复点之前，具体步骤如下：

1. 在 `test1` 这个 namespace 中产生一个名为 `demo3-restore-gcs` 的 `Restore` CR，并指定恢复到 `2022-10-10T17:21:00+08:00`：

```
kubectl apply -f restore-point-gcs.yaml
```

`restore-point-gcs.yaml` 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-gcs
  namespace: test1
spec:
  restoreMode: pitr
  br:
    cluster: demo3
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder-pitr
  pitrRestoredTs: "2022-10-10T17:21:00+08:00"
  pitrFullBackupStorageProvider:
```

```
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-full-backup-folder-pitr
```

在配置 restore-point-gcs.yaml 文件时，请参考以下信息：

- spec.restoreMode：在进行 PITR 恢复时，需要设置值为 pitr。默认值为 snapshot，即进行全量恢复。

2. 查看恢复的状态，等待恢复操作完成：

```
kubectl get jobs -n test1
```

```
NAME                       COMPLETIONS  ...
restore-demo3-restore-gcs  1/1          ...
```

也可通过以下命令查看恢复的状态：

```
kubectl get restore -n test1 -o wide
```

```
NAME               STATUS    ...
demo3-restore-gcs  Complete  ...
```

7.4.5.2.3 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

7.4.6 使用 Azure Blob Storage

7.4.6.1 使用 BR 备份 TiDB 集群数据到 Azure Blob Storage

本文介绍如何将运行在 Kubernetes 环境中的 TiDB 集群数据备份到 Azure Blob Storage 上。其中包括以下两种备份方式：

- 快照备份。使用快照备份，你可以通过全量恢复将 TiDB 集群恢复到快照备份的时刻点。
- 日志备份。使用快照备份与日志备份，你可以通过快照备份与日志备份产生的备份数据将 TiDB 集群恢复到历史任意时刻点，即Point-in-Time Recovery (PITR)。

本文使用的备份方式基于 TiDB Operator 的 Custom Resource Definition(CRD) 实现，底层使用 BR 获取集群数据，然后再将数据上传到 Azure Blob Storage 上。BR 全称为 Backup & Restore，是 TiDB 分布式备份恢复的命令行工具，用于对 TiDB 集群进行数据备份和恢复。

7.4.6.1.1 使用场景

如果你对数据备份有以下要求，可考虑使用 BR 的快照备份方式将 TiDB 集群数据以Ad-hoc 备份或定时快照备份的方式备份至 Azure Blob Storage 上：

- 需要备份的数据量较大（大于 1 TiB），而且要求备份速度较快
- 需要直接备份数据的 SST 文件（键值对）

如果你对数据备份有以下要求，可考虑使用 BR 的日志备份方式将 TiDB 集群数据以Ad-hoc 备份的方式备份至 Azure Blob Storage 上（同时也需要配合快照备份的数据，来更高效地恢复数据）：

- 需要在新集群上恢复备份集群的历史任意时刻点快照（PITR）
- 数据的 RPO 在分钟级别

如有其他备份需求，请参考备份与恢复简介选择合适的备份方式。

> 注意：
>
> - 快照备份只支持 TiDB v3.1 及以上版本。
> - 日志备份只支持 TiDB v6.3 及以上版本。
> - 使用 BR 备份出的数据只能恢复到 TiDB 数据库中，无法恢复到其他数据库中。

7.4.6.1.2 Ad-hoc 备份

Ad-hoc 备份支持快照备份，也支持启动和停止日志备份任务，以及清理日志备份数据等操作。

要进行 Ad-hoc 备份，你需要创建一个自定义的 `Backup` Custom Resource (CR) 对象来描述本次备份。创建好 `Backup` 对象后，TiDB Operator 根据这个对象自动完成具体的备份过程。如果备份过程中出现错误，程序不会自动重试，此时需要手动处理。

本文假设对部署在 Kubernetes `test1` 这个 namespace 中的 TiDB 集群 `demo1` 进行数据备份。下面是具体的操作过程。

前置条件：准备 Ad-hoc 备份环境

1. 创建备份需要的 RBAC 相关资源：

```
kubectl apply -n test1 -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
```

```
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]
---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
EOF
```

2. 参考Azure 账号授权授予远程存储访问权限。Azure 提供两种方式进行授权。授权成功后，`test1` namespace 中应存在名为 `azblob-secret` 或 `azblob-secret-ad` 的 Secret 对象。

> **注意：**
>
> - 授权账户应至少具备对 Blob 数据的写入权限，例如具备参与者角色。
> - 在创建 Secret 对象时，你可以自定义其名称。为便于说明，本文统一使用 `azblob-secret` 作为示例 Secret 对象名称。

快照备份

执行以下步骤，进行快照备份：

在 test1 这个 namespace 中创建一个名为 demo1-full-backup-azblob 的 Backup CR，用于快照备份：

```
kubectl apply -f full-backup-azblob.yaml
```

full-backup-azblob.yaml 文件内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-azblob
  namespace: test1
spec:
  backupType: full
  br:
    cluster: demo1
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
    # options:
    # - --lastbackupts=420134118382108673
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-full-backup-folder
    #accessTier: Hot
```

在配置 full-backup-azblob.yaml 文件时，请参考以下信息：

- 如果需要增量备份，只需要在 spec.br.options 中指定上一次的备份时间戳 --lastbackupts 即可。有关增量备份的限制，可参考使用 BR 进行备份与恢复。
- 关于 Azure Blob Storage 相关配置，请参考Azure Blob Storage 存储字段介绍。
- .spec.br 中的一些参数是可选的，例如 logLevel、statusAddr 等。完整的 .spec.br 字段的详细解释，请参考BR 字段介绍。
- .spec.azblob.secretName：填写你在创建 Secret 对象时设置的名称，例如 azblob-secret。
- 更多 Backup CR 字段的详细解释参考Backup CR 字段介绍。

查看快照备份的状态

创建好 Backup CR 后，TiDB Operator 会根据 Backup CR 自动开始备份。你可以通过如下命令查看备份状态：

```
kubectl get backup -n test1 -o wide
```

从上述命令的输出中，你可以找到描述名为 demo1-full-backup-azblob 的 Backup CR 的如下信息，其中 COMMITTS 表示快照备份的时刻点：

```
NAME                      TYPE   MODE      STATUS    BACKUPPATH
    ↪                                  COMMITTS           ...
demo1-full-backup-azblob full snapshot Complete  azure://my-container/my-
    ↪ full-backup-folder/ 436979621972148225 ...
```

日志备份

你可以使用一个 Backup CR 来描述日志备份任务的启动、停止以及清理日志备份数据等操作。本节示例创建了名为 demo1-log-backup-azblob 的 Backup CR。具体操作如下所示。

logSubcommand 字段说明

在 Backup 自定义资源 (CR) 中，你可以使用 logSubcommand 字段控制日志备份任务的状态。logSubcommand 支持以下三个命令：

- log-start：该命令用于启动新的日志备份任务，或恢复已暂停的任务。使用此命令可以开始日志备份流程，或从暂停状态恢复任务。
- log-pause：该命令用于暂停当前正在进行的日志备份任务。暂停任务后，你可以使用 log-start 命令恢复任务。
- log-stop：该命令用于永久停止日志备份任务。执行此命令后，Backup CR 会进入停止状态，且无法再次启动。

这些命令提供了对日志备份任务生命周期的精细控制，支持启动、暂停、恢复和停止操作，帮助有效管理 Kubernetes 环境中的日志数据保留。

启动日志备份

1. 在 test1 这个 namespace 中创建一个名为 demo1-log-backup-azblob 的 Backup CR。

   ```
   kubectl apply -f log-backup-azblob.yaml
   ```

   log-backup-azblob.yaml 文件内容如下：

   ```
   apiVersion: br.pingcap.com/v1alpha1
   kind: Backup
   metadata:
     name: demo1-log-backup-azblob
     namespace: test1
   spec:
   ```

```
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
    #accessTier: Hot
```

2. 等待启动操作完成：

```
kubectl get jobs -n test1
```

```
NAME                                        COMPLETIONS  ...
backup-demo1-log-backup-azblob-log-start 1/1             ...
```

3. 查看新增的 Backup CR：

```
kubectl get backup -n test1
```

```
NAME                       MODE   STATUS   ....
demo1-log-backup-azblob log    Running ....
```

查看日志备份的状态

通过查看 Backup CR 的信息，可查看日志备份的状态。

```
kubectl describe backup -n test1
```

从上述命令的输出中，你可以找到描述名为 demo1-log-backup-azblob 的 Backup CR 的如下信息，其中 Log Checkpoint Ts 表示日志备份可恢复的最近时间点：

```
Status:
Backup Path: azure://my-container/my-log-backup-folder/
Commit Ts:  436568622965194754
Conditions:
    Last Transition Time: 2022-10-10T04:45:20Z
    Status:              True
    Type:                Scheduled
    Last Transition Time: 2022-10-10T04:45:31Z
    Status:              True
    Type:                Prepare
    Last Transition Time: 2022-10-10T04:45:31Z
    Status:              True
```

```
    Type:                Running
Log Checkpoint Ts:    436569119308644661
```

暂停日志备份

你可以通过将 Backup 自定义资源 (CR) 的 logSubcommand 字段设置为 log-pause 来暂停日志备份任务。下面以暂停启动日志备份中创建的名为 demo1-log-backup-azblob 的 CR 为例。

```
kubectl edit backup demo1-log-backup-azblob -n test1
```

要暂停日志备份任务，只需将 logSubcommand 的值从 log-start 修改为 log-pause，然后保存并退出编辑器。

```
kubectl apply -f log-backup-azblob.yaml
```

修改后 log-backup-azblob.yaml 文件内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-azblob 的 Backup CR 的 STATUS 从 Running 变成了 Pause：

```
kubectl get backup -n test1
```

```
NAME                      MODE    STATUS    ....
demo1-log-backup-azblob log     Pause     ....
```

恢复日志备份

如果日志备份任务已暂停，你可以通过将 logSubcommand 字段设置为 log-start 来恢复该任务。下面以恢复暂停日志备份中已暂停的 demo1-log-backup-azblob CR 为例。

> **Note:**
>
> 此操作仅适用于处于暂停状态 (Pause) 的任务，无法恢复状态为 Fail 或 Stopped 的任务。

```
kubectl edit backup demo1-log-backup-azblob -n test1
```

要恢复日志备份任务，只需将 logSubcommand 的值从 log-pause 更改为 log-start，然后保存并退出编辑器。修改后的内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
```

可以看到名为 demo1-log-backup-azblob 的 Backup CR 的 STATUS 从 Paused 状态变为 Running：

```
kubectl get backup -n test1
```

```
NAME                          MODE    STATUS    ....
demo1-log-backup-azblob       log     Running   ....
```

停止日志备份

你可以通过将 Backup 自定义资源 (CR) 的 logSubcommand 字段设置为 log-stop 来停止日志备份。下面以停止启动日志备份中创建的名为 demo1-log-backup-azblob 的 CR 为例。

```
kubectl edit backup demo1-log-backup-azblob -n test1
```

将 logSubcommand 的值修改为 log-stop，然后保存并退出编辑器。修改后的内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
    #accessTier: Hot
```

可以看到名为 demo1-log-backup-azblob 的 Backup CR 的 STATUS 从 Running 变成了 Stopped：

```
kubectl get backup -n test1
```

```
NAME                      MODE   STATUS    ....
demo1-log-backup-azblob log    Stopped   ....
```

Stopped 是日志备份的终止状态。在此状态下，无法再次更改备份状态，但你仍然可以清理日志备份数据。

清理日志备份数据

1. 由于你在开启日志备份的时候已经创建了名为 demo1-log-backup-azblob 的 Backup CR，因此可以直接更新该 Backup CR 的配置，来激活清理日志备份数据的操作。执行如下操作来清理 2022-10-10T15:21:00+08:00 之前的所有日志备份数据。

   ```
   kubectl edit backup demo1-log-backup-azblob -n test1
   ```

   在最后新增一行字段 spec.logTruncateUntil: "2022-10-10T15:21:00+08:00"，保存并退出。更新后的内容如下：

   ```
   apiVersion: br.pingcap.com/v1alpha1
   kind: Backup
   metadata:
     name: demo1-log-backup-azblob
     namespace: test1
   spec:
     backupMode: log
   ```

```
  logSubcommand: log-start/log-pause/log-stop
  br:
    cluster: demo1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
    #accessTier: Hot
  logTruncateUntil: "2022-10-10T15:21:00+08:00"
```

2. 等待清理操作完成：

```
kubectl get jobs -n test1
```

```
NAME                                          COMPLETIONS  ...
...
backup-demo1-log-backup-azblob-log-truncate 1/1      ...
```

3. 查看 Backup CR 的信息：

```
kubectl describe backup -n test1
```

```
...
Log Success Truncate Until: 2022-10-10T15:21:00+08:00
...
```

也可以通过以下命令查看：

```
kubectl get backup -n test1 -o wide
```

```
NAME              MODE      STATUS    ...  LOGTRUNCATEUNTIL
demo1-log-backup-azblob log       Complete  ...  2022-10-10T15
    ↪ :21:00+08:00
```

压缩日志备份

对于 TiDB v9.0.0 及以上版本的集群，你可以使用 CompactBackup CR 将日志备份数据压缩为 SST 格式，以加速下游的日志恢复 (Point-in-time recovery, PITR)。

本节基于前文的日志备份示例，介绍如何使用压缩日志备份。

在 test1 namespace 中创建一个名为 demo1-compact-backup 的 CompactBackup CR。

```
kubectl apply -f compact-backup-demo1.yaml
```

compact-backup-demo1.yaml 的内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: CompactBackup
metadata:
  name: demo1-compact-backup
  namespace: test1
spec:
  startTs: "***"
  endTs: "***"
  concurrency: 8
  maxRetryTimes: 2
  br:
    cluster: demo1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
```

其中，startTs 和 endTs 指定 demo1-compact-backup 需要压缩的日志备份时间范围。任何包含至少一个该时间区间内写入的日志都会被送去压缩。因此，最终的压缩结果可能包含该时间范围之外的写入数据。

azblob 设置应与需要压缩的日志备份的存储设置相同，CompactBackup 会读取相应地址的日志文件并进行压缩。

查看压缩日志备份状态

创建 CompactBackup CR 后，TiDB Operator 会自动开始压缩日志备份。你可以运行以下命令查看备份状态：

```
kubectl get cpbk -n test1
```

从上述命令的输出中，你可以找到描述名为 demo1-compact-backup 的 CompactBackup CR 的信息，输出示例如下：

```
NAME                    STATUS                PROGRESS
    ↪                                    MESSAGE
demo1-compact-backup Complete [READ_META(17/17),COMPACT_WORK(1291/1291)]
```

如果 STATUS 字段显示为 Complete 则代表压缩日志备份已经完成。

备份示例

备份全部集群数据

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
```

```
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    prefix: my-folder
```

备份单个数据库的数据

以下示例中，备份 `db1` 数据库的数据。

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    sendCredToTikv: false
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    prefix: my-folder
```

备份单张表的数据

以下示例中，备份 `db1.table1` 表的数据。

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
```

```
  - "db1.table1"
  br:
    cluster: demo1
    sendCredToTikv: false
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    prefix: my-folder
```

使用表库过滤功能备份多张表的数据

以下示例中，备份 db1.table1 表和 db1.table2 表的数据。

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  # ...
  br:
    cluster: demo1
    sendCredToTikv: false
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    prefix: my-folder
```

7.4.6.1.3 定时快照备份

你可以通过设置备份策略来对 TiDB 集群进行定时备份，同时设置备份的保留策略以避免产生过多的备份。定时快照备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次快照备份，定时快照备份底层通过 Ad-hoc 快照备份来实现。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

执行快照备份

依据准备 Ad-hoc 备份环境时所选择的远程存储访问授权方式，你需要使用下面对应的方法将数据定时备份到 Azure Blob Storage 上：

- 方法 1：如果通过了访问密钥的方式授权，你可以按照以下说明创建 BackupSchedule CR，开启 TiDB 集群定时快照备份：

```
kubectl apply -f backup-scheduler-azblob.yaml
```

backup-scheduler-azblob.yaml 文件内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-azblob
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    br:
      cluster: demo1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
      # sendCredToTikv: true
   azblob:
     secretName: azblob-secret-ad
     container: my-container
     prefix: my-folder
```

- 方法 2：如果通过了 Azure AD 的方式授权，你可以按照以下说明创建 BackupSchedule CR，开启 TiDB 集群定时快照备份：

```
kubectl apply -f backup-scheduler-azblob.yaml
```

backup-scheduler-azblob.yaml 文件内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-azblob
  namespace: test1
spec:
  #maxBackups: 5
```

```
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    br:
      cluster: demo1
      sendCredToTikv: false
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    azblob:
      secretName: azblob-secret-ad
      container: my-container
      prefix: my-folder
```

从以上 backup-scheduler-azblob.yaml 文件配置示例可知，backupSchedule 的配置由两部分组成。一部分是 backupSchedule 独有的配置，另一部分是 backupTemplate。

- 关于 backupSchedule 独有的配置项具体介绍，请参考BackupSchedule CR 字段介绍。
- backupTemplate 用于指定集群及远程存储相关的配置，字段和 Backup CR 中的 spec 一样，详细介绍可参考Backup CR 字段介绍。

定时快照备份创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

查看定时快照备份下面所有的备份条目：

```
kubectl get backup -l tidb.pingcap.com/backup-schedule=demo1-backup-schedule
    ↪ -azblob -n test1
```

7.4.6.1.4 集成管理定时快照备份和日志备份

BackupSchedule CR 可以集成管理 TiDB 集群的定时快照备份和日志备份，通过设置备份的保留时间可以定期回收快照备份和日志备份，且能保证在保留期内可以通过快照备份和日志备份进行 PITR 恢复。

本节示例创建了名为 integrated-backup-schedule-azblob 的 BackupSchedule CR，其中访问 Azure Blob Storage 的方式参考Azure 账号授权。具体操作如下所示。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

创建 BackupSchedule

1. 在 test1 这个 namespace 中创建一个名为 integrated-backup-schedule-azblob 的 BackupSchedule CR。

```
kubectl apply -f integrated-backup-scheduler-azblob.yaml
```

integrated-backup-scheduler-azblob.yaml 文件内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-azblob
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: schedule-backup-folder-snapshot
      #accessTier: Hot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: schedule-backup-folder-log
      #accessTier: Hot
```

以上 integrated-backup-scheduler-azblob.yaml 文件配置示例中，backupSchedule 的配置由三部分组成：backupSchedule 独有的配置，快照备份配置 backupTemplate，日志备份配置 logBackupTemplate。

关于 backupSchedule 配置项具体介绍，请参考BackupSchedule CR 字段介绍。

2. backupSchedule 创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

日志备份会随着 backupSchedule 创建，可以通过如下命令查看 backupSchedule 的 status.logBackup，即日志备份名称。

```
kubectl describe bks integrated-backup-schedule-azblob -n test1
```

3. 在进行集群恢复时，需要指定备份的路径。你可以通过如下命令查看定时快照备份下面所有的备份条目，在命令输出中 MODE 为 snapshot 的条目为快照备份，MODE 为 log 的条目为日志备份。

```
kubectl get bk -l tidb.pingcap.com/backup-schedule=integrated-backup-
    ↪ schedule-azblob -n test1
```

```
NAME                                                    MODE       STATUS
    ↪ ....
integrated-backup-schedule-azblob-2023-03-08t02-48-00 snapshot Complete
    ↪  ....
log-integrated-backup-schedule-azblob                   log        Running
    ↪ ....
```

7.4.6.1.5 集成定时快照备份、日志备份和压缩日志备份

为了加快下游恢复速度，可以在 BackupSchedule CR 中添加压缩日志备份。压缩日志备份可以定期压缩远程存储中的日志备份文件。你必须先启用日志备份，才能使用压缩日志备份。本节基于上一节内容进行扩展。

前置条件：准备定时快照备份环境

同准备 Ad-hoc 备份环境。

创建 BackupSchedule

1. 在 test1 这个 namespace 中创建一个名为 integrated-backup-schedule-azblob 的 BackupSchedule CR。

```
kubectl apply -f integrated-backup-scheduler-azblob.yaml
```

integrated-backup-scheduler-azblob.yaml 文件内容如下：

```
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-azblob
  namespace: test1
spec:
```

```
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  compactInterval: "1h"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: schedule-backup-folder-snapshot
      #accessTier: Hot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: schedule-backup-folder-log
      #accessTier: Hot
```

以上 integrated-backup-scheduler-azblob.yaml 文件配置示例中，backupSchedule 配置基于上一节内容，新增了 compactBackup 相关设置，主要改动如下：

- 新增 BackupSchedule.spec.compactInterval 字段，用于指定日志压缩备份的时间间隔。建议不要超过定时快照备份的间隔，并控制在定时快照备份间隔的二分之一至三分之一之间。
- 新增 BackupSchedule.spec.compactBackupTemplate 字段。请确保 BackupSchedule.spec.compactBackupTemplate.azblob 配置与 BackupSchedule.spec.logBackupTemplate.azblob 保持一致。

关于 backupSchedule 配置项具体介绍，请参考BackupSchedule CR 字段介绍。

2. backupSchedule 创建完成后，可以通过以下命令查看定时快照备份的状态：

```
kubectl get bks -n test1 -o wide
```

压缩日志备份会随着 backupSchedule 创建，可以通过如下命令查看 CompactBackup CR 的信息。

```
kubectl get cpbk -n test1
```

7.4.6.1.6 删除备份的 Backup CR

如果你不再需要已备份的 Backup CR，请参考删除备份的 Backup CR。

7.4.6.1.7 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

7.4.6.2 使用 BR 恢复 Azure Blob Storage 上的备份数据

本文介绍如何将 Azure Blob Storage 上的备份数据恢复到 Kubernetes 环境中的 TiDB 集群，其中包括以下两种恢复方式：

- 全量恢复，可以将 TiDB 集群恢复到快照备份的时刻点。备份数据来自于快照备份。
- PITR 恢复，可以将 TiDB 集群恢复到历史任意时刻点。备份数据来自于快照备份和日志备份。

本文使用的恢复方式基于 TiDB Operator 的 Custom Resource Definition (CRD) 实现，底层使用 BR 进行数据恢复。BR 全称为 Backup & Restore，是 TiDB 分布式备份恢复的命令行工具，用于对 TiDB 集群进行数据备份和恢复。

PITR 全称为 Point-in-time recovery，该功能可以让你在新集群上恢复备份集群的历史任意时刻点的快照。使用 PITR 功能恢复时需要快照备份数据和日志备份数据。在恢复时，首先将快照备份的数据恢复到 TiDB 集群中，再以快照备份的时刻点作为起始时刻点，并指定任意恢复时刻点，将日志备份数据恢复到 TiDB 集群中。

> 注意：
>
> - BR 只支持 TiDB v3.1 及以上版本。
> - BR 的 PITR 恢复功能只支持 TiDB v6.3 及以上版本。
> - BR 恢复的数据无法被同步到下游，因为 BR 直接导入 SST/LOG 文件，而下游集群目前没有办法获得上游的 SST/LOG 文件。

7.4.6.2.1 全量恢复

本节示例将存储在 Azure Blob Storage 上指定路径 `spec.azblob.container` 存储桶中 `spec.azblob.prefix` 文件夹下的快照备份数据恢复到 namespace `test2` 中的 TiDB 集群 `demo2`。以下是具体的操作过程。

前置条件：完成数据备份

本节假设 Azure Blob Storage 中的桶 `my-container` 中文件夹 `my-full-backup-folder` 下存储着快照备份产生的备份数据。关于如何备份数据，请参考使用 BR 备份 TiDB 集群数据到 Azure Blob Storage。

第 1 步：准备恢复环境

使用 BR 将 Azure Blob Storage 上的备份数据恢复到 TiDB 前，请按照以下步骤准备恢复环境。

1. 创建恢复需要的 RBAC 相关资源：

```
kubectl apply -n test2 -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]
---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
EOF
```

2. 参考Azure 账号授权授予远程存储访问权限。Azure 提供两种方式进行授权。授权成功后，namespace 中应存在名为 `azblob-secret` 或 `azblob-secret-ad` 的 Secret 对象。

注意：

- 授权账户应至少具备对 Blob 数据的写入权限，例如具备参与者角色。
- 在创建 Secret 对象时，你可以自定义其名称。为便于说明，本文统一使用 azblob-secret 作为示例 Secret 对象名称。

第 2 步：将指定备份数据恢复到 TiDB 集群

在 test2 这个 namespace 中创建一个名为 demo2-restore-azblob 的 Restore CR，用于恢复快照备份产生的数据：

```
kubectl apply -n test2 -f restore-full-azblob.yaml
```

restore-full-azblob.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-azblob
  namespace: test2
spec:
  br:
    cluster: demo2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-full-backup-folder
```

在配置 restore-full-azblob.yaml 文件时，请参考以下信息：

- 关于 Azure Blob Storage 相关配置，请参考Azure Blob Storage 存储字段介绍。
- .spec.br 中的一些参数为可选项，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释，请参考BR 字段介绍。
- .spec.azblob.secretName：填写你在创建 Secret 对象时设置的名称，例如 azblob-secret。

- 更多 Restore CR 字段的详细解释，请参考Restore CR 字段介绍。

创建好 Restore CR 后，可通过以下命令查看恢复的状态：

```
kubectl get restore -n test2 -o wide
```

```
NAME                  STATUS    ...
demo2-restore-azblob Complete ...
```

7.4.6.2.2 PITR 恢复

本节示例在 namespace test3 中的 TiDB 集群 demo3 上执行 PITR 恢复，分为以下两步：

1. 使用 spec.pitrFullBackupStorageProvider.azblob.container 存储桶中 spec.pitrFullBackupStorageProvider.azblob.prefix 文件夹下的快照备份数据，将集群恢复到快照备份的时刻点。
2. 使用 spec.azblob.container 存储桶中 spec.azblob.prefix 文件夹下的日志备份的增量数据，将集群恢复到备份集群的历史任意时刻点。

下面是具体的操作过程。

前置条件：完成数据备份

本节假设 Azure Blob Storage 中的桶 my-container 中存在两份备份数据，分别是：

- 在日志备份期间，进行快照备份产生的备份数据，存储在 my-full-backup-folder-pitr 文件夹下。
- 日志备份产生的备份数据，存储在 my-log-backup-folder-pitr 文件夹下。

关于如何备份数据，请参考使用 BR 备份 TiDB 集群数据到 Azure Blob Storage。

> 注意：
>
> 指定的恢复时间点需要在快照备份时刻点之后，日志备份 checkpoint-ts 之前。

第 1 步：准备恢复环境

与全量恢复的准备步骤相同。

第 2 步：将指定备份数据恢复到 TiDB 集群

本节示例中首先将快照备份恢复到集群中，因此 PITR 的恢复时刻点需要在快照备份的时刻点之后，并在日志备份的最新恢复点之前。具体步骤如下：

1. 在 test3 这个 namespace 中创建一个名为 demo3-restore-azblob 的 Restore CR，并指定恢复到 2022-10-10T17:21:00+08:00：

```
kubectl apply -n test3 -f restore-point-azblob.yaml
```

restore-point-azblob.yaml 文件内容如下：

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-azblob
  namespace: test3
spec:
  restoreMode: pitr
  br:
    cluster: demo3
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder-pitr
  pitrRestoredTs: "2022-10-10T17:21:00+08:00"
  pitrFullBackupStorageProvider:
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: my-full-backup-folder-pitr
```

在配置 restore-point-azblob.yaml 文件时，请参考以下信息：

- spec.restoreMode：在进行 PITR 恢复时，需要设置值为 pitr。默认值为 snapshot，即进行全量恢复。

2. 查看恢复的状态，等待恢复操作完成：

```
kubectl get jobs -n test3
```

```
NAME                          COMPLETIONS  ...
restore-demo3-restore-azblob 1/1          ...
```

也可通过以下命令查看恢复的状态：

```
kubectl get restore -n test3 -o wide
```

```
NAME                  STATUS    ...
demo3-restore-azblob Complete ...
```

7.4.6.2.3 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

7.5 运维

7.5.1 查看日志

本文档介绍如何查看 TiDB 集群各组件日志，以及 TiDB 慢查询日志。

7.5.1.1 TiDB 集群各组件日志

通过 TiDB Operator 部署的 TiDB 各组件默认将日志输出在容器的 `stdout` 和 `stderr` 中。可以通过下面的方法查看单个 Pod 的日志：

```
kubectl logs -n ${namespace} ${pod_name}
```

如果这个 Pod 由多个 Container 组成，可以查看这个 Pod 内某个 Container 的日志：

```
kubectl logs -n ${namespace} ${pod_name} -c ${container_name}
```

请通过 `kubectl logs --help` 获取更多查看 Pod 日志的方法。

7.5.1.2 TiDB 组件慢查询日志

TiDB 3.0 及以上的版本中，慢查询日志和应用日志区分开，可以通过名为 `slowlog` 的 sidecar 容器查看慢查询日志：

```
kubectl logs -n ${namespace} ${pod_name} -c slowlog
```

> **注意：**
>
> 慢查询日志的格式与 MySQL 的慢查询日志相同，但由于 TiDB 自身的特点，其中的一些具体字段可能存在差异，因此解析 MySQL 慢查询日志的工具不一定能完全兼容 TiDB 的慢查询日志。

7.5.2 暂停同步 Kubernetes 上的 TiDB 集群

本文介绍如何通过配置暂停同步 Kubernetes 上的 TiDB 集群。

7.5.2.1 TiDB Operator 的同步机制

在 TiDB Operator 中，控制器会不断对比 Custom Resource (CR) 对象中记录的期望状态与实际状态，并通过创建、更新或删除 Kubernetes 资源来驱动 TiDB 集群满足期望状态。这个不断调整的过程通常被称为同步。更多细节参见TiDB Operator 架构。

7.5.2.2 暂停同步的应用场景

暂停同步适用于以下场景：

- 避免意外的滚动升级

  为防止 TiDB Operator 新版本的兼容性问题影响集群，升级 TiDB Operator 之前，可以先暂停同步集群。升级 TiDB Operator 之后，逐个恢复同步集群或者在指定时间恢复同步集群，以此来观察 TiDB Operator 版本升级对集群的影响。

- 避免多次滚动重启集群

  在某些情况下，一段时间内可能会多次修改 TiDB 集群配置，但是又不想多次滚动重启集群。为了避免多次滚动重启集群，可以先暂停同步集群，在此期间，对 TiDB 集群的任何配置都不会生效。集群配置修改完成后，恢复集群同步，此时暂停同步期间的所有配置修改都能在一次重启过程中被应用。

- 维护时间窗口

  在某些情况下，只允许在特定时间窗口内滚动升级或重启 TiDB 集群。因此可以在维护时间窗口之外的时间段暂停 TiDB 集群的同步过程，这样在维护时间窗口之外对 TiDB 集群的任何配置都不会生效；在维护时间窗口内，可以通过恢复 TiDB 集群同步来允许滚动升级或者重启 TiDB 集群。

7.5.2.3 暂停同步 TiDB 集群

如果想要暂停同步 TiDB 集群，可以在 Cluster CR 中配置 `spec.paused: true`。

1. 使用以下命令修改集群配置，其中 `${cluster_name}` 表示 TiDB 集群名称，`${namespace}` 表示 TiDB 集群所在的 namespace。

```
kubectl patch cluster ${cluster_name} -n ${namespace} --type merge -p
    ↪ '{"spec":{"paused": true}}'
```

2. TiDB 集群同步暂停后，可以使用以下命令查看 TiDB Operator Pod 日志，确认 TiDB 集群同步状态。其中 `${pod_name}` 表示 TiDB Operator Pod 的名称，`${namespace}` 表示 TiDB Operator 所在的 namespace。

```
kubectl logs ${pod_name} -n ${namespace} | grep paused
```

   输出类似如下内容时，表示 TiDB 集群同步已经暂停。

```
2025-04-25T09:27:27.866Z INFO   TiCDC   cluster paused is updating {"
    ↪ from": false, "to": true}
2025-04-25T09:27:27.866Z INFO   TiDB    cluster paused is updating {"
    ↪ from": false, "to": true}
2025-04-25T09:27:27.867Z INFO   TiFlash cluster paused is updating {"
    ↪ from": false, "to": true}
```

```
2025-04-25T09:27:27.868Z INFO   PD      cluster paused is updating {"
    ↪ from": false, "to": true}
2025-04-25T09:27:27.868Z INFO   TiKV    cluster paused is updating {"
    ↪ from": false, "to": true}
```

7.5.2.4 恢复同步 TiDB 集群

如果想要恢复 TiDB 集群的同步，可以在 Cluster CR 中配置 `spec.paused: false`。恢复同步后，TiDB Operator 会立即开始处理暂停期间累积的所有配置变更。

1. 使用以下命令修改集群配置，其中 `${cluster_name}` 表示 TiDB 集群名称，`${namespace}` 表示 TiDB 集群所在的 namespace。

```
kubectl patch cluster ${cluster_name} -n ${namespace} --type merge -p
    ↪ '{"spec":{"paused": false}}'
```

2. 恢复 TiDB 集群同步后，可以使用以下命令查看 TiDB Operator Pod 日志，确认 TiDB 集群同步状态。其中 `${pod_name}` 表示 TiDB Operator Pod 的名称，`${namespace}` 表示 TiDB Operator 所在的 namespace。

```
kubectl logs ${pod_name} -n ${namespace} | grep "paused"
```

输出结果示例如下，可以看到同步成功时间戳（`cluster paused` 状态从 `true` 变为 `false` 的时间戳）大于暂停同步日志中显示的时间戳（`cluster paused` 状态从 `false` 变为 `true` 的时间戳），表示 TiDB 集群同步已经被恢复。

```
2025-04-25T09:32:38.867Z INFO   TiKV    cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO   PD      cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO   TiFlash cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO   TiCDC   cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO   TiDB    cluster paused is updating {"
    ↪ from": true, "to": false}
```

7.5.3 挂起和恢复 Kubernetes 上的 TiDB 集群

本文介绍如何通过配置 `Cluster` 对象来挂起和恢复 Kubernetes 上的 TiDB 集群。挂起操作会停止集群中所有组件的 Pod，但不会删除 `Cluster` 对象及其相关资源（例如 Service、PVC 等），从而保留集群的数据和配置，便于后续恢复。

7.5.3.1 使用场景

挂起 TiDB 集群适用于以下场景：

- 临时释放测试环境中的计算资源
- 停止长期不使用的开发集群
- 临时停止集群但保留数据和配置

7.5.3.2 注意事项

挂起 TiDB 集群前，需要注意以下事项：

- 挂起操作将中断集群服务
- 已有的连接会被强制断开
- PVC 和数据仍会保留并占用存储空间
- 集群相关的 Service 和配置保持不变

7.5.3.3 挂起 TiDB 集群

如果你需要挂起 TiDB 集群，执行以下步骤：

1. 在 `Cluster` 对象中，将 `spec.suspendAction.suspendCompute` 字段设置为 `true`，以挂起整个 TiDB 集群：

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  suspendAction:
    suspendCompute: true
  # ...
```

2. 挂起 TiDB 集群后，通过以下命令观察到 TiDB 集群的 Pod 逐步被删除：

```
kubectl -n ${namespace} get pods -w
```

7.5.3.4 恢复 TiDB 集群

在 TiDB 集群被挂起后，如果需要恢复 TiDB 集群，执行以下步骤：

1. 在 `Cluster` 对象中，将 `spec.suspendAction.suspendCompute` 字段设置为 `false`，以恢复被挂起的整个 TiDB 集群：

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  suspendAction:
    suspendCompute: false
  # ...
```

2. 恢复 TiDB 集群后，通过以下命令观察到 TiDB 集群的 Pod 逐步被创建：

```
kubectl -n ${namespace} get pods -w
```

7.5.4 重启 Kubernetes 上的 TiDB 集群

在使用 TiDB 集群的过程中，如果某个 Pod 存在内存泄漏等问题，可能需要重启集群。本文介绍如何优雅滚动重启 TiDB 集群中某个组件的所有 Pod，或单独优雅重启某个 Pod。

> **警告：**
>
> 在生产环境中，强烈建议不要强制删除 TiDB 集群的 Pod。虽然 TiDB Operator 会自动重新创建被删除的 Pod，但这仍可能会导致部分访问 TiDB 集群的请求失败。

7.5.4.1 优雅滚动重启某个组件的所有 Pod

要优雅滚动重启某个组件（例如 PD、TiKV 或 TiDB）的所有 Pod，需要修改该组件对应的 Component Group Custom Resource (CR) 配置，在 `.spec.template.metadata` 部分添加 `pingcap.com/restartedAt` 的 label 或 annotation，并将其值设置为一个保证幂等性的字符串，例如当前时间。

以下示例展示如何为 PD 组件添加一个 annotation，从而触发对该 `PDGroup` 下所有 PD Pod 的优雅滚动重启：

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  replicas: 3
  template:
```

```
  metadata:
    annotations:
      pingcap.com/restartedAt: 2025-06-30T12:00
```

7.5.4.2 优雅重启某个组件的单个 Pod

你可以单独重启 TiDB 集群中的特定 Pod。不同组件的 Pod，操作略有不同。

对于 TiKV Pod，为确保有足够时间驱逐 Region leader，在删除 Pod 时需要指定 `--grace-period` 选项，否则操作可能失败。以下示例为 TiKV Pod 设置了 60 秒的宽限期：

```
kubectl -n ${namespace} delete pod ${pod_name} --grace-period=60
```

其他组件的 Pod 可以直接删除，TiDB Operator 会自动优雅重启这些 Pod：

```
kubectl -n ${namespace} delete pod ${pod_name}
```

7.5.5 销毁 Kubernetes 上的 TiDB 集群

本文描述了如何销毁 Kubernetes 集群上的 TiDB 集群。

7.5.5.1 销毁使用 `Cluster` 管理的 TiDB 集群

要销毁使用 `Cluster` 管理的 TiDB 集群，执行以下命令：

```
kubectl delete cluster ${cluster_name} -n ${namespace}
```

8 故障诊断

8.1 Kubernetes 上的 TiDB 常见部署错误

本文介绍了 Kubernetes 上 TiDB 常见部署错误以及处理办法。

8.1.1 Pod 未正常创建

创建备份恢复任务后，如果 Pod 没有创建，则可以通过以下方式进行诊断：

```
kubectl get backups -n ${namespace}
kubectl get jobs -n ${namespace}
kubectl describe backups -n ${namespace} ${backup_name}
kubectl describe jobs -n ${namespace} ${backupjob_name}
kubectl describe restores -n ${namespace} ${restore_name}
```

8.1.2 Pod 处于 Pending 状态

Pod 处于 Pending 状态，通常都是资源不满足导致的，比如：

- 使用持久化存储的 PD、TiKV、TiFlash、Backup、Restore Pod 使用的 PVC 的 StorageClass 不存在或 PV 不足
- Kubernetes 集群中没有节点能满足 Pod 申请的 CPU 或内存
- TiDB、TiProxy 等组件使用的证书没有配置

此时，可以通过 `kubectl describe pod` 命令查看 Pending 的具体原因：

```
kubectl describe po -n ${namespace} ${pod_name}
```

8.1.2.1 CPU 或内存资源不足

如果是 CPU 或内存资源不足，可以通过降低对应组件的 CPU 或内存资源申请，使其能够得到调度，或是增加新的 Kubernetes 节点。

8.1.2.2 PVC 的 StorageClass 不存在

如果是 PVC 的 StorageClass 找不到，可采取以下步骤：

1. 通过以下命令获取集群中可用的 StorageClass：

   ```
   kubectl get storageclass
   ```

2. 将 `storageClassName` 修改为集群中可用的 StorageClass 名字。

3. 使用下述方式更新配置文件：

   如果是运行 backup/restore 的备份/恢复任务，首先需要运行 `kubectl delete bk ↪ ${backup_name} -n ${namespace}` 删掉老的备份/恢复任务，再运行 `kubectl ↪ apply -f backup.yaml` 重新创建新的备份/恢复任务。

4. 删除对应的 PVC：

   ```
   kubectl delete pvc -n ${namespace} ${pvc_name}
   ```

8.1.2.3 可用 PV 不足

如果集群中有 StorageClass，但可用的 PV 不足，则需要添加对应的 PV 资源。

8.1.3 Pod 处于 CrashLoopBackOff 状态

Pod 处于 CrashLoopBackOff 状态意味着 Pod 内的容器重复地异常退出（异常退出后，容器被 Kubelet 重启，重启后又异常退出，如此往复）。定位方法有很多种。

8.1.3.1 查看 Pod 内当前容器的日志

```
kubectl -n ${namespace} logs -f ${pod_name}
```

8.1.3.2 查看 Pod 内容器上次启动时的日志信息

```
kubectl -n ${namespace} logs -p ${pod_name}
```

确认日志中的错误信息后，可以根据 tidb-server 启动报错、tikv-server 启动报错、pd-server 启动报错中的指引信息进行进一步排查解决。

8.1.3.3 ulimit 不足

另外，TiKV 在 ulimit 不足时也会发生启动失败的状况，对于这种情况，可以修改 Kubernetes 节点的 `/etc/security/limits.conf` 调大 ulimit：

```
root       soft       nofile       1000000
root       hard       nofile       1000000
root       soft       core         unlimited
root       soft       stack        10240
```

8.2 Kubernetes 上的 TiDB 集群常见异常

本文介绍 TiDB 集群运行过程中常见异常以及处理办法。

8.2.1 TiDB 长连接被异常中断

许多负载均衡器 (Load Balancer) 会设置连接空闲超时时间。当连接上没有数据传输的时间超过设定值，负载均衡器会主动将连接中断。若发现 TiDB 使用过程中，长查询会被异常中断，可检查客户端与 TiDB 服务端之间的中间件程序。若其连接空闲超时时间较短，可尝试增大该超时时间。若不可修改，可打开 TiDB `tcp-keep-alive` 选项，启用 TCP keepalive 特性。

默认情况下，Linux 发送 keepalive 探测包的等待时间为 7200 秒。若需减少该时间，可通过 `podSecurityContext` 字段配置 `sysctls`。

- 如果 Kubernetes 集群内的 kubelet 允许配置 `--allowed-unsafe-sysctls=net.*`，请使用Overlay 功能按如下方式配置 TiDBGroup：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
spec:
  template:
    spec:
      overlay:
```

```
      pod:
        spec:
          securityContext:
            sysctls:
              - name: net.ipv4.tcp_keepalive_time
                value: "300"
```

- 如果 Kubernetes 集群内的 kubelet 不允许配置 `--allowed-unsafe-sysctls=net.*`，请使用Overlay 功能按如下方式配置 TiDBGroup：

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            initContainers:
              - name: init
                image: busybox
                commands:
                  - "sh"
                  - "-c"
                  - "sysctl"
                  - "-w"
                  - "net.ipv4.tcp_keepalive_time=300"
                securityContext:
                  privileged: true
```

9 参考

9.1 架构

9.1.1 TiDB Operator 架构

本文档介绍 TiDB Operator 的架构及其工作原理。

9.1.1.1 架构

下图是 TiDB Operator 的架构概览。

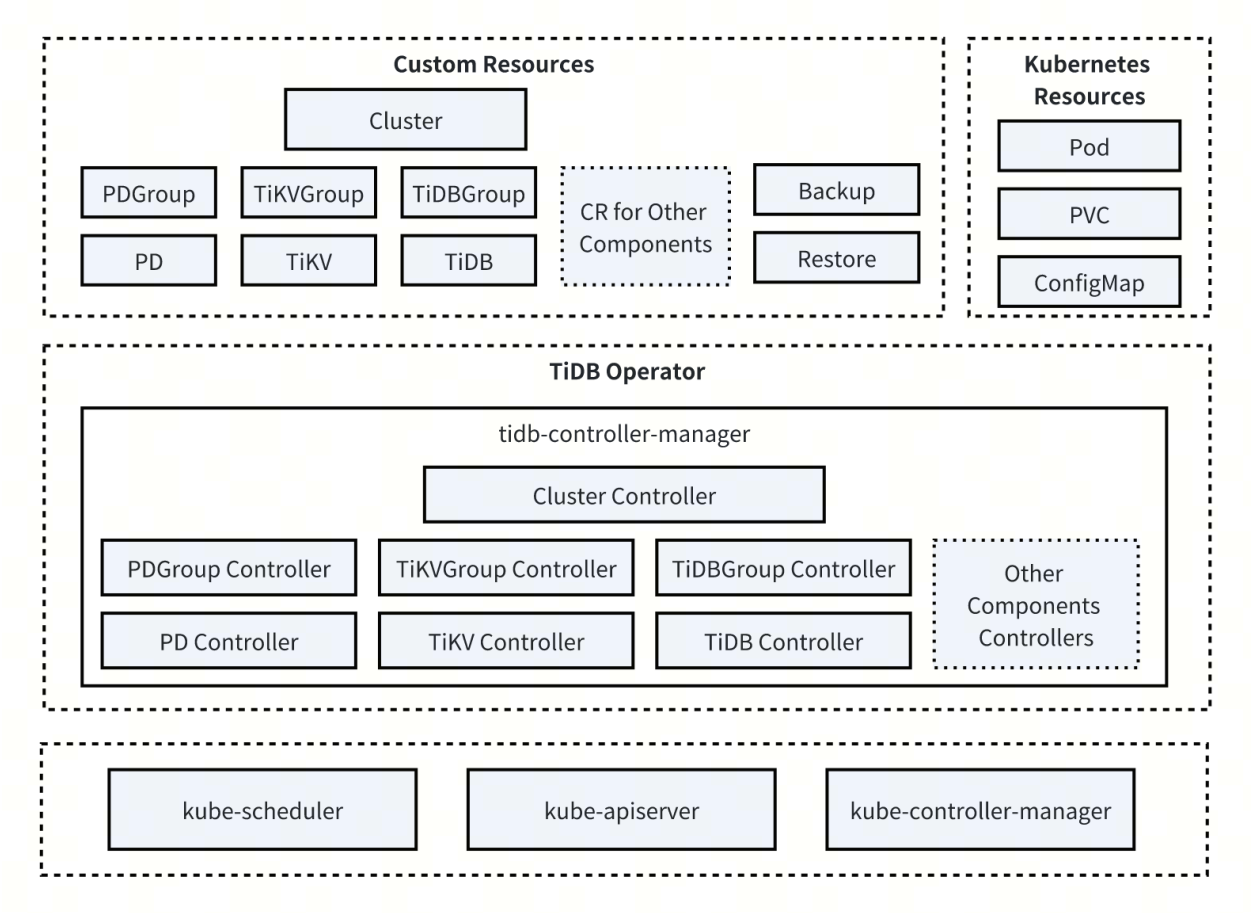

Figure 2: TiDB Operator Architecture

图中包含多个由 Custom Resource Definition (CRD) 定义的资源对象，例如 `Cluster`、`PDGroup`、`PD`、`TiKVGroup`、`TiKV`、`TiDBGroup`、`TiDB`、`Backup`、`Restore` 等。部分资源的说明如下：

- `Cluster`：表示一个完整的 TiDB 集群，它包含了 TiDB 集群的一些通用配置和功能开关，并反映集群的整体状态。该 CRD 被设计为 TiDB 集群的“命名空间”，TiDB 集群的所有组件都必须引用一个 `Cluster` CR。
- `ComponentGroup`：用于描述一组具有相同配置的 TiDB 集群组件，例如：
  - `PDGroup` 表示一组具有相同配置的 PD 实例
  - `TiKVGroup` 表示一组具有相同配置的 TiKV 实例
  - `TiDBGroup` 表示一组具有相同配置的 TiDB 实例
- `Component`：用于描述一个 TiDB 集群组件，例如：
  - `PD` 表示一个 PD 实例

  - `TiKV` 表示一个 TiKV 实例
  - `TiDB` 表示一个 TiDB 实例

- `Backup`：用于描述用户期望执行的 TiDB 集群备份任务。
- `Restore`：用于描述用户期望执行的 TiDB 集群恢复任务。

9.1.1.2 流程解析

TiDB Operator 采用声明式 API，通过监控用户定义的资源对象实现自动化控制。其核心流程如下：

1. 用户通过 `kubectl` 创建 `Cluster` 以及其他组件的自定义资源 (Custom Resource, CR) 对象，例如 `PDGroup`、`TiKVGroup` 和 `TiDBGroup` 等。
2. TiDB Operator 持续监控 (Watch) 这些 CR，根据集群实际状态动态调整各组件对应的 `Pod`、`PVC` 和 `ConfigMap` 等对象。

通过这种控制（协调）循环，TiDB Operator 能够自动进行集群节点健康检查和故障恢复。部署、升级、扩缩容等操作也可以通过修改 `Cluster` 和其他组件的 CR 对象“一键”完成。

以下是以 TiKV 为例的控制流程图：

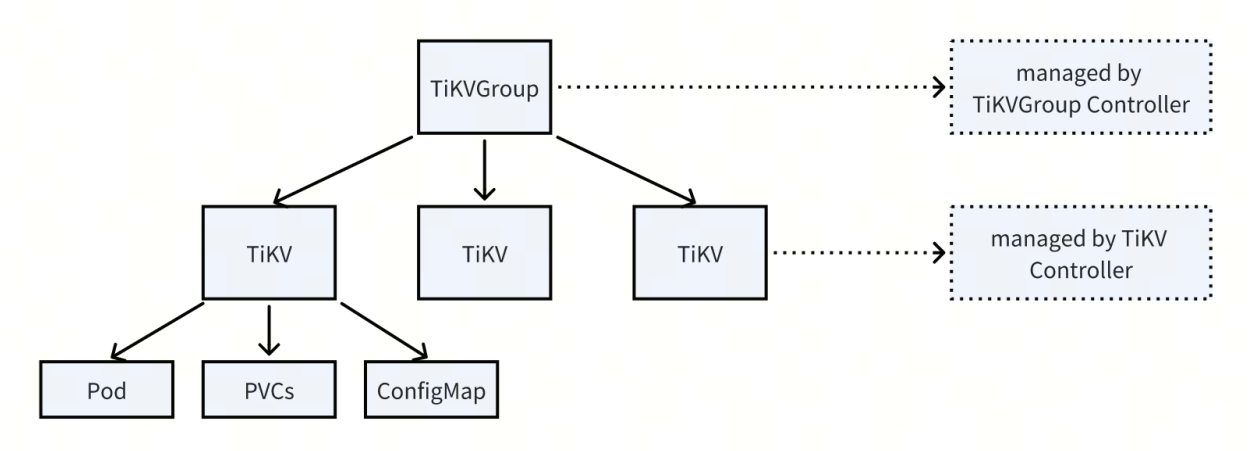

Figure 3: TiDB Operator Control Flow

在该流程中：

- TiKVGroup Controller：监听 `TiKVGroup` CR，并根据 CR 中的配置创建或更新对应的 `TiKV` CR。
- TiKV Controller：监听 `TiKV` CR，并根据 CR 中的配置创建或更新 TiKV 相关的 Pod、PVC 和 ConfigMap 等资源。

9.2 TiDB Operator v2 和 v1 的对比

由于 Kubernetes 和 TiDB 生态的快速发展，TiDB Operator v1 现有的架构和实现遇到了一些挑战。为了更好地适配 Kubernetes 和 TiDB 生态，TiDB Operator v2 对 TiDB Operator v1 进行了大幅重构。

9.2.1 TiDB Operator v2 的核心变更

9.2.1.1 拆分 `TidbCluster` CRD

最初 TiDB 集群只有 3 个核心组件：PD、TiKV、TiDB。为了尽可能简化部署，降低用户心智负担，最初的设计将所有 TiDB 集群的组件定义在了同一个 CRD `TidbCluster` 中。然而，随着 TiDB 的发展，这种设计迎来了一些挑战。

- TiDB 集群的组件不断增加，目前已经有 8 个组件定义在 `TidbCluster` CRD 中
- 为了实现状态展示，所有节点的状态都定义在了 `TidbCluster` CRD 中
- 缺乏原生对异构集群对支持，只能通过引入额外的 `TidbCluster` CR 实现异构集群
- 无法支持 `/scale` API，无法与 Kubernetes 的 HorizontalPodAutoscaler (HPA) 生态集成
- 一个巨大的 CR/CRD 可能带来难以解决的性能问题

为解决上述问题，TiDB Operator v2 将 `TidbCluster` CRD 按组件拆分为多个独立的 CRD。

9.2.1.2 移除 StatefulSet 依赖，直接管理 Pod

由于 TiDB 集群本身的复杂性，Kubernetes 原生的 Deployment 和 StatefulSet 控制器无法完美适配 TiDB 的部署和运维需求。TiDB Operator v1 通过 StatefulSet 管理所有 TiDB 组件，然而 StatefulSet 的一些限制无法最大化 Kubernetes 能够提供的能力，比如：

- StatefulSet 限制了 `VolumeClaimTemplate` 的修改，无法原生支持扩容
- StatefulSet 限制了缩容和滚动更新的顺序，导致 leader 的反复调度
- StatefulSet 限制了同一个控制器下的 Pod 配置必须相同，不得不通过复杂的启动脚本来差异化同一组 Pod 的启动参数
- 没有 API 提供 Raft member 的定义，导致重启 Pod 和移除 Raft member 的语义冲突，没有直观的方法可以移除某一个 TiKV 节点

TiDB Operator v2 移除了对 StatefulSet 的依赖，并引入了以下 CRD：

- `Cluster`
- `ComponentGroup`
- `Instance`

这三层 CRD 可以直接管理 Pod。TiDB Operator v2 通过 `ComponentGroup` CRD 来管理具有共同特性的节点，降低复杂度，通过 `Instance` CRD 来方便对单个有状态实例进行管理，提供实例级别的运维操作，保证了灵活性。

这带来了如下好处：

- 能够更好的支持 Volume 的变更
- 能够支持更合理的滚动更新顺序，比如最后重启 leader，防止 leader 的反复迁移
- 能够支持非核心组件（例如 log tail 和 istio）的原地升级，降低 TiDB Operator 升级以及 infra 变更对 TiDB 集群的影响
- 能够通过 `kubectl delete ${pod}` 优雅重启 Pod，也能通过 `kubectl delete ${instance}` 重建特定的 TiKV 节点
- 更加直观的状态展示

9.2.1.3 引入 Overlay 机制，不再直接管理 TiDB 无关的 Kubernetes 字段

每个 Kubernetes 的新版本都可能会引入一些用户需要的新字段，然而这些字段可能 TiDB Operator 并不关心。TiDB Operator v1 的开发过程中花了大量的时间在支持快速发展的 Kubernetes 的新功能，包括手动在 `TidbCluster` CRD 中添加新字段，并将新字段层层下发。TiDB Operator v2 引入了 Overlay 机制，通过统一的方式支持所有 Kubernetes 资源（尤其是 Pod）上的新字段，详情见Overlay。

9.2.1.4 其他 TiDB Operator v2 的新特性

9.2.1.4.1 增强验证能力

TiDB Operator v2 通过合法性检查规则 (Validation Rule) 和验证准入策略 (Validating Admission Policy) 增强配置校验能力，提高了系统的易用性与健壮性。

9.2.1.4.2 支持 `/status` 和 `/scale` 子资源

TiDB Operator v2 支持 CRD 子资源，可与 Kubernetes 提供的 HPA 集成，实现自动化扩缩容。

9.2.1.4.3 移除 `tidb scheduler` 组件并支持 Evenly Spread Policy

TiDB Operator v2 支持配置 Evenly Spread Policy 来将组件按需均匀分布到不同的 Region 和 Zone 上，并移除了 `tidb scheduler` 组件。

9.2.2 TiDB Operator v2 暂不支持的组件和功能

9.2.2.1 组件

9.2.2.1.1 `Binlog` (Pump + Drainer)

`Binlog` 组件已经废弃，详见 TiDB Binlog 简介。

9.2.2.1.2 Dumpling + TiDB Lightning

TiDB Operator 不再提供对 Dumpling 和 TiDB Lightning 的直接支持，建议使用 Kubernetes 原生 Job 的方式运行。

9.2.2.1.3 `TidbInitializer`

TiDB Operator v2 不再支持 `TidbInitializer` CRD，你可以使用 BootstrapSQL 的方式运行初始化的 SQL。

9.2.2.1.4 `TidbMonitor`

TiDB Operator v2 不再支持 `TidbMonitor` CRD。由于用户的监控系统通常比较复杂并且方案众多，`TidbMonitor` 往往无法很好地集成到生产级别的监控系统中。TiDB 通过更灵活的方式直接为你提供集成常用监控系统的方案，不再通过 CRD 的方式运行 Prometheus + Grafana + Alert-Manager 的组合。详情请参阅TiDB 集群的监控与告警。

9.2.2.1.5 `TidbNgMonitoring`

暂不支持 `TidbNgMonitoring`。

9.2.2.1.6 `TidbDashboard`

暂不支持通过 CRD 部署 `TidbDashboard`。你可以使用内置的 Dashboard 或者通过 Deployment 自行部署。

9.2.2.2 功能

9.2.2.2.1 跨 Namespace 部署

考虑到跨 Namespace 可能带来的安全性问题以及尚不明确的用户场景，暂不支持。

9.2.2.2.2 跨 Kubernetes 集群部署

考虑到跨 Kubernetes 集群可能带来的安全性问题以及尚不明确的用户场景，暂不支持。

9.2.2.2.3 基于 EBS 卷快照的备份恢复

基于 EBS 卷快照的备份存在以下难以解决的问题：

- 成本过高：EBS 卷快照的成本非常高。
- **RTO** 过长：从 EBS 卷快照恢复的时间非常长。

随着持续的优化，TiDB BR 的性能提升显著，基于 EBS 卷快照的备份恢复不再是必需的。因此，TiDB Operator v2 不再支持该功能。

9.3 工具

9.3.1 Kubernetes 上的 TiDB 工具指南

本文介绍如何在 Kubernetes 环境中使用 TiDB 相关的工具，包括 PD Control、TiKV Control 和 TiDB Control。

9.3.1.1 在 Kubernetes 上使用 PD Control

PD Control 是 PD 的命令行工具。在使用 PD Control 操作 Kubernetes 上的 TiDB 集群时，需要先使用 `kubectl port-forward` 建立本地到 PD 服务的连接：

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd 2379:2379 &>/
    ↪ tmp/portforward-pd.log &
```

执行上述命令后，就可以通过 127.0.0.1:2379 访问 PD 服务，从而直接使用 pd-ctl 命令的默认参数执行操作。例如查看 PD 配置：

```
pd-ctl -d config show
```

如果本地端口 2379 被占用，你可以指定其他端口：

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd ${local_port
    ↪ }:2379 &>/tmp/portforward-pd.log &
```

此时，需要在 pd-ctl 命令中显式指定 PD 端口：

```
pd-ctl -u 127.0.0.1:${local_port} -d config show
```

9.3.1.2 在 Kubernetes 上使用 TiKV Control

TiKV Control 是 TiKV 的命令行工具。在使用 TiKV Control 操作 Kubernetes 上的 TiDB 集群时，需要先使用 `kubectl port-forward` 建立本地到 PD 服务以及目标 TiKV 节点的连接。

以下示例将本地 2379 端口转发至 PD 服务：

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd 2379:2379 &>/
    ↪ tmp/portforward-pd.log &
```

以下示例将本地 20160 端口转发至目标 TiKV Pod：

```
kubectl port-forward -n ${namespace} ${pod_name} 20160:20160 &>/tmp/
    ↪ portforward-tikv.log &
```

建立连接后，即可通过本地的对应端口访问 PD 服务和 TiKV 节点：

```
tikv-ctl --host 127.0.0.1:20160 --pd 127.0.0.1:2379 ${subcommands}
```

9.3.1.3 在 Kubernetes 上使用 TiDB Control

TiDB Control 是 TiDB 的命令行工具，使用 TiDB Control 时，需要从本地访问 TiDB 节点和 PD 服务，因此建议使用 `kubectl port-forward` 建立本地到集群中 TiDB 节点和 PD 服务的连接：

以下示例将本地 2379 端口转发至 PD 服务：

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd 2379:2379 &>/
    ↪ tmp/portforward-pd.log &
```

以下示例将本地 10080 端口转发至 TiDB Pod：

```
kubectl port-forward -n ${namespace} ${pod_name} 10080:10080 &>/tmp/
    ↪ portforward-tidb.log &
```

建立连接后，即可使用 `tidb-ctl` 执行相关操作。例如，查看 `mysql` 数据库的 schema：

```
tidb-ctl schema in mysql
```

10 版本发布历史

10.1 v2.0

10.1.1 TiDB Operator 2.0.0-beta.0 Release Notes

发布日期：2025 年 7 月 9 日

TiDB Operator 版本：2.0.0-beta.0

随着 TiDB 和 Kubernetes 生态的快速发展，TiDB Operator 发布 v2.0.0-beta.0 版本，对 v1 进行了全面重构，旨在提供更稳定、高效且易于维护的集群管理体验。

关于 TiDB Operator v2 与 v1 的详细差异，请参考 TiDB Operator v2 与 v1 版本对比。

> **警告：**
>
> 此版本为 beta 版本，建议在生产环境中部署前进行充分测试。

10.1.1.1 主要变化和改进

10.1.1.1.1 核心架构重构

TiDB Operator v2 对 v1 的核心架构进行了全面重构，主要包括：

- **CRD 拆分**：将 v1 中的 `TidbCluster` CRD 拆分为多个独立的 CRD，实现更细粒度的组件管理，提高可维护性和灵活性。
- **直接管理 Pod**：移除对 StatefulSet 的依赖，改为直接管理 Pod，提供更高的灵活性，便于更精细地控制 Pod 的生命周期和调度行为。
- 控制器架构升级：基于 controller-runtime 框架实现控制器逻辑，简化控制器的开发流程，提升开发效率，并增强系统的稳定性与可靠性。

10.1.1.1.2 新特性与功能增强

- 支持 **Overlay** 字段：
  - 允许用户在不修改 TiDB Operator 源码的情况下，灵活地为 Pod 指定 Kubernetes 支持的所有字段
  - 提供安全校验机制，防止关键系统标签被误覆盖
- 拓扑感知调度：
  - 支持 `EvenlySpread` 策略，实现 Pod 在不同拓扑域间的均匀分布
  - 支持拓扑权重配置，可灵活控制各拓扑域中实例的分布比例
  - 提升集群高可用性和容错能力
- 增强字段校验：
  - 集成 Kubernetes 的合法性检查规则 (Validation Rule) 和验证准入策略 (Validating Admission Policy)
  - 支持字段格式与取值范围校验
  - 提供更明确、易理解的错误提示信息，便于问题定位
- 支持 **CRD 子资源**：
  - 支持 `status` 子资源，实现统一的状态管理
  - 支持 `scale` 子资源，可与 HorizontalPodAutoscaler (HPA) 集成，实现自动扩缩容
  - 增强与 Kubernetes 生态系统的集成能力
- 优化配置管理：
  - 优化配置哈希算法，避免因无效变更导致不必要的滚动更新

10.1.1.1.3 移除功能

- 移除基于 AWS EBS 卷快照的备份恢复相关功能。
- 移除 `tidb-scheduler` 组件。
- 移除 `TiDBInitializer`、`TiDBDashboard`、`DMCluster`、`FedVolumeBackup`、`FedVolumeBackupSchedule` ↪ 、`FedVolumeRestore` 等 CRD。
- 移除 `TiDBMonitor`、`TiDBNGMonitoring` 等 CRD，相关功能已通过其他方式集成，详情请查阅 TiDB 集群的监控与告警。

10.1.1.2 致谢

感谢所有为 TiDB Operator 做出贡献的开发者和社区成员！我们期待您的反馈和建议，共同完善这个重要的里程碑版本。
