TiDB in Kubernetes 用户文档

PingCAP Inc.

20230330

Table of Contents

1 TiDB in 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私有部署的 Kubernetes 集群上。

TiDB Operator 与适用的 TiDB 版本的对应关系如下：

TiDB Operator 版本	适用的 TiDB 版本
dev	dev
TiDB >= 5.4	1.3
5.1 <= TiDB < 5.4	1.3（推荐），1.2
3.0 <= TiDB < 5.1	1.3（推荐），1.2，1.1
2.1 <= TiDB < v3.0	1.0（停止维护）

2.1.1 使用 TiDB Operator 管理 TiDB 集群

TiDB Operator 提供了多种方式来部署 Kubernetes 上的 TiDB 集群：

- 测试环境：
  - kind
  - Minikube
  - Google Cloud Shell
- 生产环境：
  - 公有云：参考AWS 部署文档，GKE 部署文档 (beta)，或阿里云部署文档在对应的公有云上一键部署生产可用的 TiDB 集群并进行后续的运维管理；
  - 现有 Kubernetes 集群：首先按照部署 TiDB Operator在集群中安装 TiDB Operator，再根据在标准 Kubernetes 集群上部署 TiDB 集群来部署你的 TiDB 集群。对于生产级 TiDB 集群，你还需要参考TiDB 集群环境要求调整 Kubernetes 集群配置并根据本地 PV 配置为你的 Kubernetes 集群配置本地 PV，以满足 TiKV 的低延迟本地存储需求。

在任何环境上部署前，都可以参考TiDB 集群配置来自定义 TiDB 配置。

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
- TiDB 集群升级
- TiDB 集群配置变更
- TiDB 集群备份与恢复
- 配置 TiDB 集群故障自动转移
- 监控 TiDB 集群
- 查看 TiDB 日志
- 维护 TiDB 所在的 Kubernetes 节点

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。

Kubernetes 上的 TiDB 提供了专用的命令行工具 `tkctl` 用于集群管理和辅助诊断，同时，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`tkctl` 使用指南 来使用 `tkctl`；
- 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。

2.2 What's New in TiDB Operator 1.1

TiDB Operator 1.1 在 1.0 基础上新增 TiDB 4.0 功能特性支持，TiKV 数据加密、TLS 证书配置等。新增 TiFlash、TiCDC 新组件部署支持，同时在易用性上做了许多改进，提供与 Kubernetes 原生资源一致的用户体验。以下是主要变化：

2.2.1 扩展性

- TidbCluster CR 支持部署管理 PD Discovery 组件，可完全替代 tidb-cluster chart 管理 TiDB 集群
- TidbCluster CR 新增 Pump、TiFlash、TiCDC、Dashboard 支持
- 新增可选的准入控制器改进升级、扩缩容体验，并提供灰度发布功能
- tidb-scheduler 支持任意维度的 HA 调度和调度器 preemption
- 使用 tikv-importer chart 部署、管理 TiKV Importer

2.2.2 易用性

- 新增 TidbMonitor CR 用于部署集群监控
- 新增 TidbInitializer CR 用于初始化集群
- 新增 Backup、BackupSchedule、Restore CR 用于备份恢复集群。备份、恢复支持 S3 和 GCS
- 优雅重启 TiDB 集群组件

2.2.3 安全性

- 支持 TiDB 集群各组件及客户端 TLS 证书配置
- 支持 TiKV 数据存储加密

2.2.4 实验性特性

- 新增 `TidbClusterAutoScaler` 实现集群自动伸缩功能（开启 `AutoScaling` 特性开关后使用）
- 新增可选的增强型 StatefulSet 控制器，提供对指定 Pod 进行删除的功能（开启 `AdvancedStatefulSet` 特性开关后使用）

完整发布日志参见 1.1 CHANGE LOG。

TiDB Operator 在 Kubernetes 上部署参见安装文档，CRD 文档参见 API References。

2.3 TiDB Operator v1.1 重要注意事项

本文介绍 TiDB Operator v1.1 版本重要注意事项。

2.3.1 PingCAP 不再继续更新维护 tidb-cluster chart

从 TiDB Operator v1.1.0 开始，PingCAP 不再继续更新 tidb-cluster chart，原来由 tidb-cluster chart 负责管理的组件或者功能在 v1.1 中的变更如下：

组件、功能	v1.1
TiDB Cluster (PD, TiDB, TiKV)	TidbCluster CR
TiDB Monitor	TidbMonitor CR
TiDB Initializer	TidbInitializer CR
Scheduled Backup	BackupSchedule CR
Pump	TidbCluster CR
Drainer	tidb-drainer chart
Importer	tikv-importer chart

- tidb-cluster chart 会继续发布，但是 PingCAP 不再添加新功能，社区贡献者仍然可

以为 tidb-cluster chart 添加新功能。
- 通过 v1.0.x TiDB Operator 部署的 TiDB 集群，在 TiDB Operator 升级到 v1.1 之后，仍然可以通过 v1.1 版本的 tidb-cluster chart 升级和管理 TiDB 集群。

2.3.2 tidb-cluster chart 管理的组件或者功能切换到 v1.1 完整支持的方式

虽然 TiDB Operator v1.1 版本仍然支持用户使用 Helm 和 tidb-cluster chart 管理集群，但是由于 tidb-cluster chart 不再新增功能，用户可以自己为 tidb-cluster chart 贡献新功能或者切换到 TiDB Operator v1.1 完整支持的方式。

下面介绍如何将 tidb-cluster chart 管理的组件或者功能切换到 v1.1 完整支持的方式。

2.3.2.1 Discovery

Discovery 服务直接由 TiDB Operator 内部生成，不再需要用户做任何配置。

2.3.2.2 PD、TiDB、TiKV

在 tidb-cluster chart 中，PD、TiDB、TiKV 配置由 Helm 渲染成 ConfigMap，从 TiDB Operator v1.1 开始，PD、TiDB、TiKV 配置也可以直接在 TiDBCluster CR 中配置，具体配置方法可以参考通过 TidbCluster 配置 TiDB 集群。

> 注意：
>
> 由于 TiDB Operator 渲染配置的方式和 Helm 渲染配置的方式不同，从 tidb-cluster chart values.yaml 中的配置迁移到 CR 中的配置，会引起对应组件滚动升级。

2.3.2.3 Monitor

可以参考TiDB 集群的监控与告警创建 TidbMonitor CR，管理 Monitor 组件。

> 注意：
>
> - TidbMonitor CR 中的 `metadata.name` 需要和集群中 TidbCluster CR 的名字保持一致。
> - 由于 TiDB Operator 渲染资源的方式和 Helm 渲染资源的方式不同，从 tidb-cluster chart values.yaml 中的配置迁移到 TidbMonitor CR，会引起 Monitor 组件滚动升级。

2.3.2.4 Initializer

- 如果在升级到 TiDB Operator v1.1 之前，初始化 Job 已经执行，初始化 Job 不需要从 tidb-cluster chart 中迁移到 TidbInitializer CR。
- 如果在升级到 TiDB Operator v1.1 之前，没有执行过初始化 Job，也没有修改过 TiDB 服务 root 用户的密码，升级到 TiDB Operator v1.1 之后，需要执行初始化，可以参考Kubernetes 上的集群初始化配置进行配置。

2.3.2.5 Pump

升级到 TiDB Operator v1.1 之后，可以修改 TidbCluster CR，添加 Pump 相关配置，通过 TidbCluster CR 管理 Pump 组件：

```
spec
  ...
  pump:
    baseImage: pingcap/tidb-binlog
    version: v5.0.6
    replicas: 1
    storageClassName: local-storage
    requests:
      storage: 30Gi
    schedulerName: default-scheduler
    config:
      addr: 0.0.0.0:8250
      gc: 7
      heartbeat-interval: 2
```

按照集群实际情况修改 `version`、`replicas`、`storageClassName`、`requests.storage` 等配置。

> **注意：**
>
> 由于 TiDB Operator 渲染资源的方式和 Helm 渲染资源的方式不同，从 tidb-cluster chart values.yaml 中的配置迁移到 TidbCluster CR，会引起 Pump 组件滚动升级。

2.3.2.6 Scheduled Backup

升级到 TiDB Operator v1.1 之后，可以通过 BackupSchedule CR 配置定时全量备份：

- 如果 TiDB 集群版本 < v3.1，可以参考Dumpling 定时全量备份

- 如果 TiDB 集群版本 >= v3.1，可以参考BR 定时全量备份

> 注意：
>
> - BackupSchedule CR Dumpling 方式目前只支持备份到 s3、gcs，BR 方式只支持备份到 s3，如果升级之前的定时全量备份是备份到本地 PVC，则升级后不能切换到 CR 方式管理。
> - 如果切换到 CR 方式管理，请删除原有定时全量备份的 Cronjob，以防止重复备份。

2.3.2.7 Drainer

- 如果在升级到 TiDB Operator v1.1 之前，没有部署 Drainer，现在需要新部署，可以参考Drainer 部署。
- 如果在升级到 TiDB Operator v1.1 之前，已经通过 `tidb-drainer` chart 部署 Drainer，继续用 `tidb-drainer` chart 管理。
- 如果在升级到 TiDB Operator v1.1 之前，已经通过 `tidb-cluster` chart 部署 Drainer，建议直接用 kubectl 管理。

2.3.2.8 TiKV Importer

- 如果在升级到 TiDB Operator v1.1 之前，没有部署 TiKV Importer，现在需要新部署，可以参考TiKV Importer 部署。
- 如果在升级到 TiDB Operator v1.1 之前，已经部署 TiKV Importer，建议直接用 kubectl 管理。

2.3.3 其他由 chart 管理的组件或者功能切换到 v1.1 支持的方式

2.3.3.1 Ad-hoc 备份

升级到 TiDB Operator v1.1 之后，可以通过 Backup CR 进行备份。Dumpling 方式只支持全量备份，BR 方式支持全量备份与增量备份：

- 如果 TiDB 集群版本 < v3.1，可以参考Dumpling Ad-hoc 全量备份
- 如果 TiDB 集群版本 >= v3.1，可以参考BR Ad-hoc 备份

> 注意：
>
> Backup CR Dumpling 方式和 BR 方式目前只支持备份到 S3、GCS。如果升级之前的 Ad-hoc 全量备份是备份到本地 PVC，则不能切换到 CR 方式管理。

2.3.3.2 备份恢复

升级到 TiDB Operator v1.1 之后，可以通过 Restore CR 进行备份恢复：

- 如果 TiDB 集群版本 < v3.1，可以参考使用 TiDB Lightning 恢复 S3 兼容存储上的备份数据。
- 如果 TiDB 集群版本 >= v3.1，可以参考使用 BR 恢复 S3 兼容存储上的备份数据。

> 注意：
>
> Restore CR Lightning 方式和 BR 方式目前只支持从 S3、GCS 获取备份数据进行恢复。如果需要从本地 PVC 获取备份数据进行恢复，则不能切换到 CR 方式管理。

3 快速上手 TiDB Operator

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，请参阅部署 > 部署 TiDB 集群章节。

基本步骤如下：

1. 创建 Kubernetes 测试集群
2. 部署 TiDB Operator
3. 部署 TiDB 集群和监控
4. 连接 TiDB 集群
5. 升级 TiDB 集群
6. 销毁 TiDB 集群

如果您已经有一个 Kubernetes 集群，可直接部署 TiDB Operator。

如果您想做生产部署，参考以下文档：

- 公有云

  - AWS 部署文档
  - GKE 部署文档 (beta)
  - 阿里云部署文档

- 自托管 Kubernetes 集群

  - 集群环境要求
  - 参考本地 PV 配置让 TiKV 使用高性能本地存储
  - 在 Kubernetes 部署 TiDB Operator
  - 在标准 Kubernetes 上部署 TiDB 集群

3.1 创建 Kubernetes 测试集群

本节介绍了两种创建 Kubernetes 测试集群的方法，可用于测试 TiDB Operator 管理的 TiDB 集群。

- 使用 kind (在 Docker 中运行 Kubernetes)
- 使用 minikube (在虚拟机中运行 Kubernetes)

您也可以使用 Google Cloud Shell 在 Google Cloud Platform 的 Google Kubernetes Engine 中部署 Kubernetes 集群，并遵循教程来部署 TiDB Operator 和 TiDB 集群：

- 打开 Google Cloud Shell

3.1.1 使用 kind 创建 Kubernetes 集群

本节介绍如何使用 kind 部署 Kubernetes 集群。

kind 是用于使用 Docker 容器作为集群节点运行本地 Kubernetes 集群的工具，是为测试本地 Kubernetes 集群而开发的。Kubernetes 集群版本取决于 kind 使用的节点镜像，您可以指定要用于节点的镜像并选择任何发布的版本。请参阅 Docker Hub 以查看可用 tags。默认使用当前 kind 支持的最新版本。

> **警告：**
>
> kind 集群仅用于测试目的，不要直接用于生产环境。

部署前，请确保满足以下要求：

- docker：版本 >= 17.03
- kubectl：版本 >= 1.12
- kind：版本 >= 0.8.0

- 若使用 Linux, net.ipv4.ip_forward 需要被设置为 1

以下以 0.8.1 版本为例：

```
kind create cluster
```

期望输出：

```
Creating cluster "kind" ...
   Ensuring node image (kindest/node:v1.18.2)
   Preparing nodes
   Writing configuration
   Starting control-plane
   Installing CNI
   Installing StorageClass
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Thanks for using kind!
```

检查集群是否创建成功：

```
kubectl cluster-info
```

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'.
```

现在就可以开始部署 TiDB Operator 了！

测试完成后，执行下面命令来销毁集群：

```
kind delete cluster
```

3.1.2 使用 minikube 创建 Kubernetes 集群

本节介绍如何使用 minikube 部署 Kubernetes 集群。

minikube 可以在虚拟机中创建一个 Kubernetes 集群，可在 macOS, Linux 和 Windows 上运行。

警告：

minikube 集群仅用于测试目的，不要直接用于生产环境。

部署前，请确保满足以下要求：

- minikube：版本 1.0.0+
  - minikube 需要安装一个兼容的 hypervisor，详情见官方安装教程。
- kubectl: 版本 >= 1.12

注意：

- 尽管 minikube 支持通过 `--vm-driver=none` 选项使用主机 Docker 而不使用虚拟机，但是目前尚没有针对 TiDB Operator 做过全面的测试，可能会无法正常工作。如果您想在不支持虚拟化的系统（例如 VPS）上试用 TiDB Operator，可以考虑使用 kind。

安装完 minikube 后，可以执行下面命令启动一个 Kubernetes 集群：

```
minikube start
```

如果一切运行正常，会看到类似下面的输出，根据操作系统和使用的 hypervisor 会有些许差异。

```
  minikube v1.10.1 on Darwin 10.15.4
  Automatically selected the hyperkit driver. Other choices: docker,
  ↪ vmwarefusion
  Downloading driver docker-machine-driver-hyperkit:
  > docker-machine-driver-hyperkit.sha256: 65 B / 65 B [---] 100.00% ? p/s
    ↪ 0s
  > docker-machine-driver-hyperkit: 10.90 MiB / 10.90 MiB 100.00% 1.76 MiB
    ↪ p
  The 'hyperkit' driver requires elevated permissions. The following
  ↪ commands will be executed:

  $ sudo chown root:wheel /Users/user/.minikube/bin/docker-machine-driver-
    ↪ hyperkit
```

```
    $ sudo chmod u+s /Users/user/.minikube/bin/docker-machine-driver-
        ↪ hyperkit


😄  Downloading VM boot image ...
    > minikube-v1.10.0.iso.sha256: 65 B / 65 B [-------------] 100.00% ? p/s
        ↪  0s
    > minikube-v1.10.0.iso: 174.99 MiB / 174.99 MiB [] 100.00% 6.63 MiB p/s
        ↪ 27s
😄  Starting control plane node minikube in cluster minikube
😄  Downloading Kubernetes v1.18.2 preload ...
    > preloaded-images-k8s-v3-v1.18.2-docker-overlay2-amd64.tar.lz4: 525.43
        ↪ MiB
😄  Creating hyperkit VM (CPUs=2, Memory=4000MB, Disk=20000MB) ...
😄  Preparing Kubernetes v1.18.2 on Docker 19.03.8 ...
😄  Verifying Kubernetes components...
😄  Enabled addons: default-storageclass, storage-provisioner
😄  Done! kubectl is now configured to use "minikube"
```

对于中国大陆用户，可以使用国内 gcr.io mirror 仓库，例如 `registry.cn-hangzhou.aliyuncs.com/google_containers`。

```
minikube start --image-repository registry.cn-hangzhou.aliyuncs.com/
    ↪ google_containers
```

或者给 Docker 配置 HTTP/HTTPS 代理。

将下面命令中的 `127.0.0.1:1086` 替换为您自己的 HTTP/HTTPS 代理地址：

```
minikube start --docker-env https_proxy=http://127.0.0.1:1086 \
    --docker-env http_proxy=http://127.0.0.1:1086
```

> **注意：**
>
> 由于 minikube（默认）通过虚拟机运行，`127.0.0.1` 指向虚拟机本身，所以在有些情况下可能需要将代理修改为您的主机的实际 IP。

参考 minikube setup 查看配置虚拟机和 Kubernetes 集群的更多选项。

你可以使用 `minikube` 的子命令 `kubectl` 来进行集群操作。要使 `kubectl` 命令生效，你需要在 shell 配置文件中添加以下别名设置命令，或者在打开一个新的 shell 后执行以下别名设置命令。

```
alias kubectl='minikube kubectl --'
```

执行以下命令检查集群状态，并确保可以通过 kubectl 访问集群:

```
kubectl cluster-info
```

期望输出：

```
Kubernetes master is running at https://192.168.64.2:8443
KubeDNS is running at https://192.168.64.2:8443/api/v1/namespaces/kube-
    ↪ system/services/kube-dns:dns/proxy

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

现在就可以开始部署 TiDB Operator 了！

测试完成后，执行下面命令来销毁集群：

```
minikube delete
```

3.2 部署 TiDB Operator

开始之前，确保以下要求已满足：

- 可以使用 kubectl 访问的 Kubernetes 集群
- 已安装 Helm 3

部署 TiDB Operator 的过程分为两步：安装 TiDB Operator CRDs、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://raw.githubusercontent.com/pingcap/tidb-operator/v1
    ↪ .1.15/manifests/crd.yaml
```

期望输出：

```
customresourcedefinition.apiextensions.k8s.io/tidbclusters.pingcap.com
    ↪ created
customresourcedefinition.apiextensions.k8s.io/backups.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/restores.pingcap.com created
customresourcedefinition.apiextensions.k8s.io/backupschedules.pingcap.com
    ↪ created
customresourcedefinition.apiextensions.k8s.io/tidbmonitors.pingcap.com
    ↪ created
```

```
customresourcedefinition.apiextensions.k8s.io/tidbinitializers.pingcap.com
    ↪ created
customresourcedefinition.apiextensions.k8s.io/tidbclusterautoscalers.pingcap
    ↪ .com created
```

3.2.2 安装 TiDB Operator

TiDB Operator 使用 Helm 3 安装。

1. 添加 PingCAP 仓库

```
helm repo add pingcap https://charts.pingcap.org/
```

期望输出：

```
"pingcap" has been added to your repositories
```

2. 为 TiDB Operator 创建一个命名空间

```
kubectl create namespace tidb-admin
```

期望输出：

```
namespace/tidb-admin created
```

3. 安装 TiDB Operator

```
helm install --namespace tidb-admin tidb-operator pingcap/tidb-operator
    ↪  --version v1.1.15
```

如果访问 Docker Hub 网速较慢，可以使用阿里云上的镜像：

```
helm install --namespace tidb-admin tidb-operator pingcap/tidb-operator
    ↪  --version v1.1.15 \
    --set operatorImage=registry.cn-beijing.aliyuncs.com/tidb/tidb-
        ↪ operator:v1.1.15 \
    --set tidbBackupManagerImage=registry.cn-beijing.aliyuncs.com/tidb/
        ↪ tidb-backup-manager:v1.1.15 \
    --set scheduler.kubeSchedulerImageName=registry.cn-hangzhou.
        ↪ aliyuncs.com/google_containers/kube-scheduler
```

期望输出：

```
NAME: tidb-operator
LAST DEPLOYED: Mon Jun 1 12:31:43 2020
NAMESPACE: tidb-admin
STATUS: deployed
```

```
REVISION: 1
TEST SUITE: None
NOTES:
Make sure tidb-operator components are running:

    kubectl get pods --namespace tidb-admin -l app.kubernetes.io/
        ↪ instance=tidb-operator
```

使用以下命令检查 TiDB Operator 组件是否运行起来：

```
kubectl get pods --namespace tidb-admin -l app.kubernetes.io/instance=tidb-
    ↪ operator
```

期望输出：

```
NAME                                       READY   STATUS    RESTARTS   AGE
tidb-controller-manager-6d8d5c6d64-b8lv4   1/1     Running   0          2m22s
tidb-scheduler-644d59b46f-4f6sb            2/2     Running   0          2m22s
```

当所有的 pods 都处于 Running 状态时，可进行下一步操作。

3.3 部署 TiDB 集群和监控

下面分别介绍 TiDB 集群和监控的部署方法。

3.3.1 部署 TiDB 集群

```
kubectl create namespace tidb-cluster && \
    kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com/
        ↪ pingcap/tidb-operator/v1.1.15/examples/basic/tidb-cluster.yaml
```

如果访问 Docker Hub 网速较慢，可以使用阿里云上的镜像：

```
kubectl create namespace tidb-cluster && \
    kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com/
        ↪ pingcap/tidb-operator/v1.1.15/examples/basic-cn/tidb-cluster.yaml
```

期望输出：

```
namespace/tidb-cluster created
tidbcluster.pingcap.com/basic created
```

注意：

如果要将 TiDB 集群部署到 ARM64 机器上，可以参考在 ARM64 机器上部署 TiDB 集群。

3.3.2 部署 TiDB 集群监控

```
kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com/pingcap/
    ↪ tidb-operator/v1.1.15/examples/basic/tidb-monitor.yaml
```

如果访问 Docker Hub 网速较慢，可以使用阿里云上的镜像：

```
kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com/pingcap/
    ↪ tidb-operator/v1.1.15/examples/basic-cn/tidb-monitor.yaml
```

期望输出：

```
tidbmonitor.pingcap.com/basic created
```

3.3.3 查看 Pod 状态

```
watch kubectl get po -n tidb-cluster
```

期望输出：

```
NAME                              READY   STATUS            RESTARTS   AGE
basic-discovery-6bb656bfd-kjkxw 1/1   Running           0          29s
basic-monitor-5fc8589c89-2mwx5 0/3    PodInitializing 0          20s
basic-pd-0                        1/1     Running           0          29s
```

等待所有组件 Pods 都启动，看到每种类型（pd、tikv 和 tidb）都处于 Running 状态时，您可以按 Ctrl+C 返回命令行，然后进行下一步：连接到 TiDB 集群。

期望输出：

```
NAME                              READY   STATUS    RESTARTS   AGE
basic-discovery-6bb656bfd-xl5pb 1/1   Running   0          9m9s
basic-monitor-5fc8589c89-gvgjj 3/3    Running   0          8m58s
basic-pd-0                        1/1     Running   0          9m8s
basic-tidb-0                      2/2     Running   0          7m14s
basic-tikv-0                      1/1     Running   0          8m13s
```

3.4 连接 TiDB 集群

由于 TiDB 支持 MySQL 传输协议及其绝大多数的语法，因此您可以直接使用 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，您需要在使用 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，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

首先，将端口从本地主机转发到 Kubernetes 中的 TiDB **Servcie**。我们先获取 tidb-cluster 命名空间中的服务列表：

```
kubectl get svc -n tidb-cluster
```

期望输出：

```
NAME                    TYPE       CLUSTER-IP     EXTERNAL-IP  PORT(S)
    ↪             AGE
basic-discovery         ClusterIP  10.101.69.5    <none>       10261/TCP
    ↪          10m
basic-grafana           ClusterIP  10.106.41.250  <none>       3000/TCP
    ↪          10m
basic-monitor-reloader ClusterIP 10.99.157.225 <none>       9089/TCP
    ↪          10m
basic-pd                ClusterIP  10.104.43.232  <none>       2379/TCP
    ↪          10m
basic-pd-peer           ClusterIP  None           <none>       2380/TCP
    ↪          10m
basic-prometheus        ClusterIP  10.106.177.227 <none>       9090/TCP
    ↪          10m
basic-tidb              ClusterIP  10.99.24.91    <none>       4000/TCP,10080/
    ↪ TCP 8m40s
basic-tidb-peer         ClusterIP  None           <none>       10080/TCP
    ↪          8m40s
basic-tikv-peer         ClusterIP  None           <none>       20160/TCP
    ↪          9m39s
```

这个例子中，TiDB **Service** 是 **basic-tidb**。使用以下命令转发本地端口到集群：

```
kubectl port-forward -n tidb-cluster svc/basic-tidb 4000 > pf4000.out &
```

命令会运行在后台，并将输出转发到文件 pf4000.out。所以我们可以继续在当前 shell 会话中执行命令。

3.4.3 连接 TiDB 服务

> **注意：**
>
> 当使用 MySQL Client 8.0 访问 TiDB 服务（TiDB 版本 < v4.0.7）时，如果用户账户有配置密码，必须显式指定 `--default-auth=mysql_native_password` 参数，因为 `mysql_native_password` 不再是默认的插件。

```
mysql -h 127.0.0.1 -P 4000 -u root
```

期望输出：

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 76
Server version: 5.7.25-TiDB-v5.0.6 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 5.7 compatible

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

以下是一些可以用来验证集群功能的命令。

- 创建 `hello_world` 表：

```
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: 2
              START_KEY: 7480000000000000FF3500000000000000F8
```

```
               END_KEY:
               TABLE_ID: 53
               DB_NAME: test
           TABLE_NAME: hello_world
               IS_INDEX: 0
               INDEX_ID: NULL
           INDEX_NAME: NULL
       EPOCH_CONF_VER: 1
           EPOCH_VERSION: 26
           WRITTEN_BYTES: 0
           READ_BYTES: 0
       APPROXIMATE_SIZE: 1
       APPROXIMATE_KEYS: 0
REPLICATIONSTATUS_STATE: NULL
REPLICATIONSTATUS_STATEID: NULL
1 row in set (0.011 sec)
```

- 查询 TiDB 版本号：

```
mysql> select tidb_version()\G
*************************** 1. row ***************************
tidb_version(): Release Version: v5.0.6
Edition: Community
Git Commit Hash: 6416f8d601472892d245b950dfd5547e857a1a33
Git Branch: heads/refs/tags/v5.0.6
UTC Build Time: 2021-12-23 12:26:47
GoVersion: go1.13
Race Enabled: false
TiKV Min Version: v3.0.0-60965b006877ca7234adaced7890d7b029ed1306
Check Table Before Drop: false
1 row in set (0.001 sec)
```

- 查询 TiKV 存储状态：

```
mysql> select * from information_schema.tikv_store_status\G
*************************** 1. row ***************************
       STORE_ID: 1
       ADDRESS: basic-tikv-0.basic-tikv-peer.tidb-cluster.svc:20160
   STORE_STATE: 0
STORE_STATE_NAME: Up
          LABEL: null
       VERSION: 5.0.6
       CAPACITY: 49.98GiB
       AVAILABLE: 28.47GiB
   LEADER_COUNT: 26
```

```
    LEADER_WEIGHT: 1
    LEADER_SCORE: 26
    LEADER_SIZE: 26
    REGION_COUNT: 26
    REGION_WEIGHT: 1
    REGION_SCORE: 4299796.044762109
    REGION_SIZE: 26
       START_TS: 2022-02-17 06:13:46
LAST_HEARTBEAT_TS: 2022-02-17 06:18:16
       UPTIME: 4m30.708704931s
1 row in set (0.003 sec)
```

- 查询 TiDB 集群基本信息：

  (该命令需要 TiDB 4.0 或以上版本，如果你部署的 TiDB 版本不支持该命令，请升级集群。)

```
mysql> select * from information_schema.cluster_info\G
*************************** 1. row ***************************
       TYPE: tidb
    INSTANCE: basic-tidb-0.basic-tidb-peer.tidb-cluster.svc:4000
STATUS_ADDRESS: basic-tidb-0.basic-tidb-peer.tidb-cluster.svc:10080
    VERSION: 5.0.6
    GIT_HASH: 6416f8d601472892d245b950dfd5547e857a1a33
    START_TIME: 2022-02-17T06:14:08Z
       UPTIME: 4m31.933720922s
    SERVER_ID: 0
*************************** 2. row ***************************
       TYPE: pd
    INSTANCE: basic-pd-0.basic-pd-peer.tidb-cluster.svc:2379
STATUS_ADDRESS: basic-pd-0.basic-pd-peer.tidb-cluster.svc:2379
    VERSION: 5.0.6
    GIT_HASH: 552c53ebd355eb657208d9130521e82a05ee009d
    START_TIME: 2022-02-17T06:13:22Z
       UPTIME: 5m17.933730718s
    SERVER_ID: 0
*************************** 3. row ***************************
       TYPE: tikv
    INSTANCE: basic-tikv-0.basic-tikv-peer.tidb-cluster.svc:20160
STATUS_ADDRESS: basic-tikv-0.basic-tikv-peer.tidb-cluster.svc:20180
    VERSION: 5.0.6
    GIT_HASH: 7fcfaf4a9dd6b245fa7b6ac26740effda57b5139
    START_TIME: 2022-02-17T06:13:46Z
       UPTIME: 4m53.933733552s
    SERVER_ID: 0
3 rows in set (0.023 sec)
```

3.4.4 访问 Grafana 面板

您可以转发 Grafana 服务端口，以便本地访问 Grafana 面板。

```
kubectl port-forward -n tidb-cluster svc/basic-grafana 3000 > pf3000.out &
```

Grafana 面板可在 kubectl 所运行的主机上通过 http://localhost:3000 访问。注意，如果您是非本机（比如 Docker 容器或远程服务器）上运行 `kubectl port-forward`，将无法在本地浏览器里通过 `localhost:3000` 访问。

默认用户名和密码都是 “admin” 。

了解更多使用 TiDB Operator 部署 TiDB 集群监控的信息，可以查阅TiDB 集群监控与告警。

3.5 升级 TiDB 集群

TiDB Operator 还可简化 TiDB 集群的滚动升级。以下展示使用 kubectl 命令行工具更新 TiDB 版本到 nightly 版本的过程。在此之前，先简要介绍一下用到的 kubectl 子命令。

`kubectl edit` 在交互式文本编辑器中打开资源，管理员可以在其中进行更改并保存。如果更改有效，它们将被提交到集群。如果更改无效，它们将会被拒绝并显示一条错误消息。请注意，目前尚不对所有字段进行验证。保存某些更改后，即使更改被接受也不一定会对集群生效。

`kubectl patch` 可直接应用补丁。Kubernetes 支持几种不同的补丁策略，每种策略有不同的功能、格式等。可参考 Kubernetes Patch 了解更多细节。

3.5.1 修改 TiDB 集群版本

执行以下命令，将 TiDB 集群升级到 nightly 版本：

```
kubectl patch tc basic -n tidb-cluster --type merge -p '{"spec": {"version":
    ↪  "release-4.0-nightly"} }'
```

期望输出：

```
tidbcluster.pingcap.com/basic patched
```

3.5.2 等待 Pods 重启

执行以下命令以了解集群升级组件时的进度。您可以看到某些 Pods 进入 `Terminating` 状态后，又回到 `ContainerCreating`，最后重新进入 `Running` 状态。

```
watch kubectl get po -n tidb-cluster
```

期望输出：

```
NAME                              READY   STATUS        RESTARTS   AGE
basic-discovery-6bb656bfd-7lbhx 1/1   Running       0          24m
basic-pd-0                        1/1     Terminating 0          5m31s
basic-tidb-0                      2/2     Running       0          2m19s
basic-tikv-0                      1/1     Running       0          4m13s
```

3.5.3 转发 TiDB 服务端口

当所有 Pods 都重启后，将看到版本号已更改。

需要注意的是，由于相关 Pods 已被销毁重建，这里需要重新设置端口转发。如果 `kubeclt port-forward` 进程仍然在运行，请结束进程后再转发端口。

```
kubectl port-forward -n tidb-cluster svc/basic-tidb 4000 > pf4000.out &
```

3.5.4 检查 TiDB 集群版本

```
mysql -h 127.0.0.1 -P 4000 -u root -e 'select tidb_version()\G'
```

期望输出：

> 注意：
>
> `release-4.0-nightly` 不是固定版本，不同时间会有不同结果

```
*************************** 1. row ***************************
tidb_version(): Release Version: v4.0.0-6-gdec49a126
Edition: Community
Git Commit Hash: dec49a12654c4f09f6fedfd2a0fb0154fc095449
Git Branch: release-4.0
UTC Build Time: 2020-06-01 10:07:32
GoVersion: go1.13
Race Enabled: false
TiKV Min Version: v3.0.0-60965b006877ca7234adaced7890d7b029ed1306
Check Table Before Drop: false
```

3.6 销毁 TiDB 集群

完成测试后，您可能希望销毁 TiDB 集群。

销毁 Kubernetes 集群的方法取决于其创建方式，您可参考前面 Kubernetes 创建文档说明。以下是销毁 TiDB 集群的步骤，并不会影响 Kubernetes 集群本身。

3.6.1 删除 TiDB Cluster

```
kubectl delete tc basic -n tidb-cluster
```

此命令中，`tc` 为 tidbclusters 的简称。

3.6.2 删除 TiDB Monitor

```
kubectl delete tidbmonitor basic -n tidb-cluster
```

3.6.3 删除 PV 数据

如果您的部署使用持久性数据存储，则删除 TiDB 集群将不会删除集群的数据。如果不再需要数据，可以运行以下命令来清理数据：

```
kubectl delete pvc -n tidb-cluster -l app.kubernetes.io/instance=basic,app.
    ↪ kubernetes.io/managed-by=tidb-operator && \
kubectl get pv -l app.kubernetes.io/namespace=tidb-cluster,app.kubernetes.io
    ↪ /managed-by=tidb-operator,app.kubernetes.io/instance=basic -o name |
    ↪ xargs -I {} kubectl patch {} -p '{"spec":{"
    ↪ persistentVolumeReclaimPolicy":"Delete"}}'
```

3.6.4 删除命名空间

为确保没有残余资源，您可以删除用于 TiDB 集群的命名空间。

```
kubectl delete namespace tidb-cluster
```

3.6.5 停止 `kubectl` 的端口转发

如果您仍在运行正在转发端口的 `kubectl` 进程，请终止它们：

```
pgrep -lfa kubectl
```

3.7 探索更多

如果你已经准备好在生产环境的 Kubernetes 上部署 TiDB 集群，可参阅以下文档：

- 在标准 Kubernetes 上部署 TiDB 集群
- 在 AWS EKS 上部署 TiDB 集群
- 在 GCP GKE 上部署 TiDB 集群
- 在阿里云上部署 TiDB 集群

4 部署

4.1 部署 TiDB 集群

4.1.1 在 AWS EKS 上部署 TiDB 集群

本文介绍了如何在 AWS EKS (Elastic Kubernetes Service) 上部署 TiDB 集群。

如果需要部署 TiDB Operator 及 TiDB 集群到自托管 Kubernetes 环境，请参考部署 TiDB Operator及部署 TiDB 集群等文档。

4.1.1.1 环境准备

部署前，请确认已完成以下环境准备：

- 安装 Helm 3：用于安装 TiDB Operator。
- 完成 AWS eksctl 入门中所有操作。

  该教程包含以下内容：

  - 安装并配置 AWS 的命令行工具 awscli
  - 安装并配置创建 Kubernetes 集群的命令行工具 eksctl
  - 安装 Kubernetes 命令行工具 kubectl

要验证 AWS CLI 的配置是否正确，请运行 `aws configure list` 命令。如果此命令的输出显示了 `access_key` 和 `secret_key` 的值，则 AWS CLI 的配置是正确的。否则，你需要重新配置 AWS CLI。

> 注意：
>
> 本文档的操作需要 AWS Access Key 至少具有 eksctl 所需最少权限和创建 Linux 堡垒机所涉及的服务权限。

4.1.1.2 推荐机型及存储

- 推荐机型：出于性能考虑，推荐：
  - PD 所在节点：c5.xlarge
  - TiDB 所在节点：c5.4xlarge
  - TiKV 或 TiFlash 所在节点：m5.4xlarge
- 推荐存储：因为 AWS 目前已经支持 EBS gp3 卷类型，建议使用 EBS gp3 卷类型。对于 gp3 配置，推荐：
  - TiKV：400 MiB/s 与 4000 IOPS
  - TiFlash：625 MiB/s 与 6000 IOPS
- 推荐 AMI 类型：Amazon Linux 2

4.1.1.3 创建 EKS 集群和节点池

根据 AWS 官方博客推荐和 EKS 最佳实践文档，由于 TiDB 集群大部分组件使用 EBS 卷作为存储，推荐在创建 EKS 的时候针对每个组件在每个可用区（至少 3 个可用区）创建一个节点池。

将以下配置存为 cluster.yaml 文件，并替换 `${clusterName}` 为自己想命名的集群名字。集群和节点组的命名规则需要与正则表达式 `[a-zA-Z][-a-zA-Z0-9]*` 相匹配，避免包含 _。

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: ${clusterName}
  region: ap-northeast-1

nodeGroups:
  - name: admin
    desiredCapacity: 1
    privateNetworking: true
    labels:
      dedicated: admin

  - name: tidb-1a
    desiredCapacity: 1
    privateNetworking: true
    availabilityZones: ["ap-northeast-1a"]
    instanceType: c5.2xlarge
    labels:
      dedicated: tidb
    taints:
      dedicated: tidb:NoSchedule
```

```
- name: tidb-1d
  desiredCapacity: 0
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  instanceType: c5.2xlarge
  labels:
    dedicated: tidb
  taints:
    dedicated: tidb:NoSchedule
- name: tidb-1c
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1c"]
  instanceType: c5.2xlarge
  labels:
    dedicated: tidb
  taints:
    dedicated: tidb:NoSchedule

- name: pd-1a
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1a"]
  instanceType: c5.xlarge
  labels:
    dedicated: pd
  taints:
    dedicated: pd:NoSchedule
- name: pd-1d
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  instanceType: c5.xlarge
  labels:
    dedicated: pd
  taints:
    dedicated: pd:NoSchedule
- name: pd-1c
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1c"]
  instanceType: c5.xlarge
  labels:
    dedicated: pd
  taints:
```

```
    dedicated: pd:NoSchedule

- name: tikv-1a
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1a"]
  instanceType: r5b.2xlarge
  labels:
    dedicated: tikv
  taints:
    dedicated: tikv:NoSchedule
- name: tikv-1d
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  instanceType: r5b.2xlarge
  labels:
    dedicated: tikv
  taints:
    dedicated: tikv:NoSchedule
- name: tikv-1c
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1c"]
  instanceType: r5b.2xlarge
  labels:
    dedicated: tikv
  taints:
    dedicated: tikv:NoSchedule
```

默认只需要两个 TiDB 节点，因此可以设置 `tidb-1d` 节点组的 `desiredCapacity` 为 `0`，后面如果需要可以随时扩容这个节点组。

执行以下命令创建集群：

```
eksctl create cluster -f cluster.yaml
```

该命令需要等待 EKS 集群创建完成，以及节点组创建完成并加入进去，耗时约 5~20 分钟。可参考 eksctl 文档了解更多集群配置选项。

> **警告：**
>
> 如果使用了 Regional Auto Scaling Group (ASG)：

- 为已经启动的 EC2 开启实例缩减保护，ASG 自身的实例缩减保护不需要打开。
- 设置 ASG 终止策略为 `NewestInstance`。

4.1.1.4 配置 StorageClass

本小节介绍如何为不同的存储类型配置 StorageClass，包括创建 EKS 集群后默认存在的 gp2 存储类型、gp3 存储类型（推荐）或其他 EBS 存储类型、以及用于模拟测试裸机部署性能的本地存储。

4.1.1.4.1 gp2

创建 EKS 集群后默认会存在一个 gp2 存储类型的 StorageClass。为了提高存储的 IO 写入性能，推荐配置 StorageClass 的 `mountOptions` 字段来设置存储挂载选项 `nodelalloc` 和 `noatime`。详情可见 TiDB 环境与系统配置检查。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
### ...
mountOptions:
- nodelalloc,noatime
```

4.1.1.4.2 gp3 存储类型（推荐）或其他 EBS 存储类型

如果不想使用默认的 gp2 存储类型，可以创建其他存储类型的 StorageClass，例如 gp3 存储类型（推荐）或者 io1 存储类型。

以下步骤以 gp3 存储类型为例说明如何创建并配置 gp3 存储类型的 StorageClass。

1. 对于 gp3 存储类型，请参考 AWS 文档在 EKS 上部署 Amazon Elastic Block Store (EBS) CSI driver。对于其他存储类型，请跳过此步骤。
2. 创建 StorageClass 定义。在 StorageClass 定义中，通过 `parameters.type` 字段指定需要的存储类型。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp3
provisioner: ebs.csi.aws.com
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
```

```
  fsType: ext4
  iops: "4000"
  throughput: "400"
mountOptions:
- nodelalloc,noatime
```

3. 在 TidbCluster 的 YAML 文件中，通过 `storageClassName` 字段指定 gp3 存储类来申请 gp3 类型的 EBS 存储。可以参考以下 TiKV 配置示例：

```
spec:
  tikv:
    baseImage: pingcap/tikv
    replicas: 3
    requests:
      storage: 100Gi
    storageClassName: gp3
```

4. 为了提高存储的 IO 写入性能，推荐配置 StorageClass 的 `mountOptions` 字段来设置存储挂载选项 `nodelalloc` 和 `noatime`。详情可见 TiDB 环境与系统配置检查。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
# ...
mountOptions:
- nodelalloc,noatime
```

如果想了解更多 EBS 存储类型选择和配置信息，请查看 AWS 官方文档 和 Storage Class 官方文档。

4.1.1.4.3 本地存储

请使用 AWS EBS 作为生产环境的存储类型。如果需要模拟测试裸机部署的性能，可以为 TiKV 节点池选择 AWS 部分实例类型提供的 NVMe SSD 本地存储卷，以提供更高的 IOPS 和更低的延迟。

> 注意：
>
> - 运行中的 TiDB 集群不能动态更换 StorageClass，可创建一个新的 TiDB 集群测试。
> - 由于 EKS 升级或其他原因造成的节点重建会导致本地盘数据会丢失，在重建前你需要提前备份 TiKV 数据，因此不建议在生产环境中使用本地盘。
> - 为了避免由于节点重建导致本地存储数据丢失，请参考 AWS 文档停止 TiKV 节点组的 `ReplaceUnhealthy` 功能。

要了解哪些 AWS 实例可提供本地存储卷，可以查看 AWS 实例类型列表。

下面以 `c5d.4xlarge` 为例说明如何为本地存储配置 StorageClass。

1. 为 TiKV 创建附带本地存储的节点组。

    1. 修改 `eksctl` 配置文件中 TiKV 节点组实例类型为 `c5d.4xlarge`：

    ```
    - name: tikv-1a
      desiredCapacity: 1
      privateNetworking: true
      availabilityZones: ["ap-northeast-1a"]
      instanceType: c5d.4xlarge
      labels:
        dedicated: tikv
      taints:
        dedicated: tikv:NoSchedule
      ...
    ```

    2. 创建附带本地存储的节点组：

    ```
    eksctl create nodegroups -f cluster.yaml
    ```

    若 TiKV 的节点组已存在，为避免名字冲突，可先删除再创建，或者修改节点组的名字。

2. 部署 local volume provisioner。

    1. 为了更方便地发现并管理本地存储，你需要安装 local-volume-provisioner 程序。
    2. 通过普通挂载方式将本地存储挂载到 `/mnt/ssd` 目录。
    3. 根据本地存储的挂载情况，修改 local-volume-provisioner.yaml 文件。
    4. 使用修改后的 `local-volume-provisioner.yaml`，部署并创建一个 `local-storage` 的 Storage Class：

    ```
    kubectl apply -f <local-volume-provisioner.yaml>
    ```

3. 使用本地存储。

    完成前面步骤后，local-volume-provisioner 即可发现集群内所有本地 NVMe SSD 盘。

在 local-volume-provisioner 发现本地盘后，当部署 TiDB 集群和监控时，请在 `tidb-cluster.yaml` 中添加 `tikv.storageClassName` 字段并设置为 `local-storage`。

4.1.1.5 部署 TiDB Operator

参考快速上手中部署 TiDB Operator，在 EKS 集群中部署 TiDB Operator。

4.1.1.6 部署 TiDB 集群和监控

下面介绍如何在 AWS EKS 上部署 TiDB 集群和监控组件。

4.1.1.6.1 创建 namespace

执行以下命令，创建 TiDB 集群安装的 namespace：

```
kubectl create namespace tidb-cluster
```

> 注意：
>
> 这里创建的 namespace 是指 Kubernetes 命名空间。本文档使用 `tidb-cluster` 为例，若使用了其他名字，修改相应的 `-n` 或 `--namespace` 参数为对应的名字即可。

4.1.1.6.2 部署 TiDB 集群和监控

首先执行以下命令，下载 TidbCluster 和 TidbMonitor CR 的配置文件。

```
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ examples/aws/tidb-cluster.yaml &&
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ examples/aws/tidb-monitor.yaml
```

如需了解更详细的配置信息或者进行自定义配置，请参考配置 TiDB 集群

> 注意：
>
> 默认情况下，`tidb-cluster.yaml` 文件中 TiDB 服务的 LoadBalancer 配置为 “internal”。这意味着 LoadBalancer 只能在 VPC 内部访问，而不能在外部访问。要通过 MySQL 协议访问 TiDB，你需要使用一个堡垒机或使用 `kubectl port-forward`。如果你想在互联网上公开访问 TiDB，并且知晓这样做的风险，你可以在 `tidb-cluster.yaml` 文件中将 LoadBalancer 从 “internal” 改为 “internet-facing”。

执行以下命令，在 EKS 集群中部署 TidbCluster 和 TidbMonitor CR。

```
kubectl apply -f tidb-cluster.yaml -n tidb-cluster && \
kubectl apply -f tidb-monitor.yaml -n tidb-cluster
```

当上述 yaml 文件被应用到 Kubernetes 集群后，TiDB Operator 会负责根据 yaml 文件描述，创建对应配置的 TiDB 集群及其监控。

> **注意：**
>
> 如果要将 TiDB 集群部署到 ARM64 机器上，可以参考在 ARM64 机器上部署 TiDB 集群。

4.1.1.6.3 查看 TiDB 集群启动状态

使用以下命令查看 TiDB 集群启动状态：

```
kubectl get pods -n tidb-cluster
```

当所有 pods 都处于 Running & Ready 状态时，则可以认为 TiDB 集群已经成功启动。如下是一个正常运行的 TiDB 集群的示例输出：

```
NAME                              READY   STATUS    RESTARTS   AGE
tidb-discovery-5cb8474d89-n8cxk   1/1     Running   0          47h
tidb-monitor-6fbcc68669-dsjlc     3/3     Running   0          47h
tidb-pd-0                         1/1     Running   0          47h
tidb-pd-1                         1/1     Running   0          46h
tidb-pd-2                         1/1     Running   0          46h
tidb-tidb-0                       2/2     Running   0          47h
tidb-tidb-1                       2/2     Running   0          46h
tidb-tikv-0                       1/1     Running   0          47h
tidb-tikv-1                       1/1     Running   0          47h
tidb-tikv-2                       1/1     Running   0          47h
```

4.1.1.7 访问数据库

创建好 TiDB 集群后，我们就可以访问数据库，进行测试和开发了。

4.1.1.7.1 准备一台堡垒机

我们为 TiDB 集群创建的是内网 LoadBalancer，因此可以在集群 VPC 内创建一台堡垒机来访问数据库。具体参考 AWS Linux 堡垒机文档在 AWS Console 上创建即可。

VPC 和 Subnet 需选择集群的 VPC 和 Subnet，在下拉框通过集群名字确认是否正确。可以通过以下命令查看集群的 VPC 和 Subnet 来验证：

```
eksctl get cluster -n ${clusterName}
```

同时需允许本机网络访问，并选择正确的 Key Pair 以便能通过 SSH 登录机器。

> **注意：**
>
> 除使用堡垒机以外，也可以使用 VPC Peering 连接现有机器到集群 VPC。若 EKS 创建于已经存在的 VPC 中，可使用 VPC 内现有机器。

4.1.1.7.2 安装 MySQL 客户端并连接

创建好堡垒机后，我们可以通过 SSH 远程连接到堡垒机，再通过 MySQL 客户端来访问 TiDB 集群。

使用 SSH 登录堡垒机：

```
ssh [-i /path/to/your/private-key.pem] ec2-user@<bastion-public-dns-name>
```

在堡垒机上安装 MySQL 客户端：

```
sudo yum install mysql -y
```

连接到 TiDB 集群：

```
mysql -h ${tidb-nlb-dnsname} -P 4000 -u root
```

其中 `${tidb-nlb-dnsname}` 为 TiDB Service 的 LoadBalancer 域名，可以通过命令 `kubectl get svc basic-tidb -n tidb-cluster` 输出中的 `EXTERNAL-IP` 字段查看。

以下为一个连接 TiDB 集群的示例：

```
$ mysql -h abfc623004ccb4cc3b363f3f37475af1-9774d22c27310bc1.elb.us-west-2.
    ↪ amazonaws.com -P 4000 -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 1189
Server version: 5.7.25-TiDB-v4.0.2 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 5.7 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)]> show status;
+--------------------+---------------------------------------+
| Variable_name    | Value                                 |
+--------------------+---------------------------------------+
| Ssl_cipher       |                                       |
| Ssl_cipher_list  |                                       |
```

```
| Ssl_verify_mode  | 0                                    |
| Ssl_version      |                                      |
| ddl_schema_version | 22                                 |
| server_id        | ed4ba88b-436a-424d-9087-977e897cf5ec |
+------------------+--------------------------------------+
6 rows in set (0.00 sec)
```

注意：

- MySQL 8.0 默认认证插件从 `mysql_native_password` 更新为 `caching_sha2_password`，因此如果使用 MySQL 8.0 客户端访问 TiDB 服务（TiDB 版本 < v4.0.7），并且用户账户有配置密码，需要显示指定 `--default-auth=mysql_native_password` 参数。
- TiDB（v4.0.2 起）默认会定期收集使用情况信息，并将这些信息分享给 PingCAP 用于改善产品。若要了解所收集的信息详情及如何禁用该行为，请参见 TiDB 遥测功能使用文档。

4.1.1.8 访问 Grafana 监控

先获取 Grafana 的 LoadBalancer 域名：

```
kubectl -n tidb-cluster get svc basic-grafana
```

示例输出：

```
$ kubectl get svc basic-grafana
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP
    ↪                                                        PORT(S)          AGE
basic-grafana LoadBalancer 10.100.199.42 a806cfe84c12a4831aa3313e792e3eed
    ↪ -1964630135.us-west-2.elb.amazonaws.com 3000:30761/TCP 121m
```

其中 `EXTERNAL-IP` 栏即为 LoadBalancer 域名。

你可以通过浏览器访问 `${grafana-lb}:3000` 地址查看 Grafana 监控指标。其中 `${grafana-lb}` 替换成前面获取的域名。

注意：

Grafana 默认用户名和密码均为 admin。

4.1.1.9 访问 TiDB Dashboard

如果想要安全地访问 TiDB Dashboard，详情可以参见访问 TiDB Dashboard。

4.1.1.10 升级 TiDB 集群

要升级 TiDB 集群，可以通过 `kubectl edit tc basic -n tidb-cluster` 命令修改 `spec.version`。

升级过程会持续一段时间，你可以通过 `kubectl get pods -n tidb-cluster --watch` 命令持续观察升级进度。

4.1.1.11 扩容 TiDB 集群

扩容前需要对相应的节点组进行扩容，以便新的实例有足够的资源运行。以下展示扩容 EKS 节点组和 TiDB 集群组件的操作。

4.1.1.11.1 扩容 EKS 节点组

TiKV 扩容需要保证在各可用区均匀扩容。以下是将集群 `${clusterName}` 的 `tikv-1a`、`tikv-1c`、`tikv-1d` 节点组扩容到 2 节点的示例：

```
eksctl scale nodegroup --cluster ${clusterName} --name tikv-1a --nodes 2 --
    ↪ nodes-min 2 --nodes-max 2
eksctl scale nodegroup --cluster ${clusterName} --name tikv-1c --nodes 2 --
    ↪ nodes-min 2 --nodes-max 2
eksctl scale nodegroup --cluster ${clusterName} --name tikv-1d --nodes 2 --
    ↪ nodes-min 2 --nodes-max 2
```

更多节点组管理可参考 eksctl 文档。

4.1.1.11.2 扩容 TiDB 组件

扩容 EKS 节点组后，可以使用命令 `kubectl edit tc basic -n tidb-cluster` 修改各组件的 `replicas` 为期望的新副本数进行扩容。

4.1.1.12 部署 TiFlash/TiCDC

TiFlash 是 TiKV 的列存扩展。

TiCDC 是一款通过拉取 TiKV 变更日志实现的 TiDB 增量数据同步工具。

这两个组件不是必选安装项，这里提供一个快速安装上手示例。

4.1.1.12.1 新增节点组

在 eksctl 的配置文件 cluster.yaml 中新增以下两项，为 TiFlash/TiCDC 各自新增一个节点组。`desiredCapacity` 决定期望的节点数，根据实际需求而定。

```
- name: tiflash-1a
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1a"]
  labels:
    dedicated: tiflash
  taints:
    dedicated: tiflash:NoSchedule
- name: tiflash-1d
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  labels:
    dedicated: tiflash
  taints:
    dedicated: tiflash:NoSchedule
- name: tiflash-1c
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1c"]
  labels:
    dedicated: tiflash
  taints:
    dedicated: tiflash:NoSchedule

- name: ticdc-1a
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1a"]
  labels:
    dedicated: ticdc
  taints:
    dedicated: ticdc:NoSchedule
- name: ticdc-1d
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  labels:
    dedicated: ticdc
  taints:
    dedicated: ticdc:NoSchedule
- name: ticdc-1c
  desiredCapacity: 1
  privateNetworking: true
```

```
  availabilityZones: ["ap-northeast-1c"]
  labels:
    dedicated: ticdc
  taints:
    dedicated: ticdc:NoSchedule
```

具体命令根据 EKS 集群创建情况而定：

- 若集群还未创建，使用 `eksctl create cluster -f cluster.yaml` 命令创建集群和节点组。
- 若集群已经创建，使用 `eksctl create nodegroup -f cluster.yaml` 命令只创建节点组（已经存在的节点组会忽略，不会重复创建）。

4.1.1.12.2 配置并部署 TiFlash/TiCDC

如果要部署 TiFlash，可以在 tidb-cluster.yaml 中配置 `spec.tiflash`，例如：

```
spec:
  ...
  tiflash:
    baseImage: pingcap/tiflash
    replicas: 1
    storageClaims:
    - resources:
        requests:
          storage: 100Gi
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: tiflash
```

其他参数可以参考TiDB 集群配置文档进行配置。

> **警告：**
>
> 由于 TiDB Operator 会按照 `storageClaims` 列表中的配置按顺序自动挂载 PV，如果需要为 TiFlash 增加磁盘，请确保只在列表原有配置末尾添加，并且不能修改列表中原有配置的顺序。

如果要部署 TiCDC，可以在 tidb-cluster.yaml 中配置 `spec.ticdc`，例如：

```
spec:
  ...
  ticdc:
    baseImage: pingcap/ticdc
    replicas: 1
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: ticdc
```

根据实际情况修改 `replicas` 等参数。

最后使用 `kubectl -n tidb-cluster apply -f tidb-cluster.yaml` 更新 TiDB 集群配置。

更多可参考 API 文档和集群配置文档完成 CR 文件配置。

4.1.2 在 GCP GKE 上部署 TiDB 集群

本文介绍了如何部署 GCP Google Kubernetes Engine (GKE) 集群，并在其中部署 TiDB 集群。

如果需要部署 TiDB Operator 及 TiDB 集群到自托管 Kubernetes 环境，请参考部署 TiDB Operator及部署 TiDB 集群等文档。

4.1.2.1 环境准备

部署前，请确认已完成以下环境准备：

- Helm 3：用于安装 TiDB Operator
- gcloud：用于创建和管理 GCP 服务的命令行工具
- 完成 GKE 快速入门 中的准备工作 (Before you begin)
  该教程包含以下内容：
  - 启用 Kubernetes API
  - 配置足够的配额等

4.1.2.2 推荐机型及存储

- 推荐机型：出于性能考虑，推荐以下机型：
  - PD 所在节点：`n2-standard-4`
  - TiDB 所在节点：`n2-standard-16`
  - TiKV 或 TiFlash 所在节点：`n2-standard-16`
- 推荐存储：推荐 TiKV 与 TiFlash 使用 pd-ssd 类型的存储。

4.1.2.3 配置 GCP 服务

```
gcloud config set core/project <gcp-project>
gcloud config set compute/region <gcp-region>
```

使用以上命令，设置好你的 GCP 项目和默认的区域。

4.1.2.4 创建 GKE 集群和节点池

1. 创建 GKE 集群和一个默认节点池：

```
gcloud container clusters create tidb --region us-east1 --machine-type
    ↪ n1-standard-4 --num-nodes=1
```

该命令创建一个区域 (regional) 集群，在该集群模式下，节点会在该区域中分别创建三个可用区 (zone)，以保障高可用。`--num-nodes=1` 参数，表示在各分区各自创建一个节点，总节点数为 3 个。生产环境推荐该集群模式。其他集群类型，可以参考 GKE 集群的类型。

以上命令集群创建在默认网络中，若希望创建在指定的网络中，通过 `--network/subnet` 参数指定。更多可查询 GKE 集群创建文档。

2. 分别为 PD、TiKV 和 TiDB 创建独立的节点池：

```
gcloud container node-pools create pd --cluster tidb --machine-type n2-
    ↪ standard-4 --num-nodes=1 \
    --node-labels=dedicated=pd --node-taints=dedicated=pd:NoSchedule
gcloud container node-pools create tikv --cluster tidb --machine-type
    ↪ n2-highmem-8 --num-nodes=1 \
    --node-labels=dedicated=tikv --node-taints=dedicated=tikv:
        ↪ NoSchedule
gcloud container node-pools create tidb --cluster tidb --machine-type
    ↪ n2-standard-8 --num-nodes=1 \
    --node-labels=dedicated=tidb --node-taints=dedicated=tidb:
        ↪ NoSchedule
```

此过程可能需要几分钟。

4.1.2.5 配置 StorageClass

创建 GKE 集群后默认会存在三个不同存储类型的 StorageClass：

- standard：`pd-standard` 存储类型（默认）
- standard-rwo：`pd-balanced` 存储类型
- premium-rwo：`pd-ssd` 存储类型（推荐）

为了提高存储的 IO 性能，推荐在 StorageClass 的 mountOptions 字段中，添加存储挂载选项 nodelalloc 和 noatime。详情可见 TiDB 环境与系统配置检查。

建议使用默认的 pd-ssd 存储类型 premium-rwo，或设置一个自定义的存储类型。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: pd-custom
provisioner: kubernetes.io/gce-pd
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
parameters:
  type: pd-ssd
mountOptions:
  - nodelalloc,noatime
```

> 注意：
>
> 默认的 pd-standard 存储类型不支持设置挂载选项 nodelalloc 和 noatime ↪ 。

4.1.2.5.1 使用本地存储

请使用区域永久性磁盘作为生产环境的存储类型。如果需要模拟测试裸机部署的性能，可以使用 GCP 部分实例类型提供的本地存储卷。可以为 TiKV 节点池选择这一类型的实例，以便提供更高的 IOPS 和低延迟。

> 注意：
>
> - 运行中的 TiDB 集群不能动态更换 StorageClass，可创建一个新的 TiDB 集群测试。
> - 由于 GKE 升级过程中节点重建会导致本地盘数据会丢失，在重建前你需要提前备份数据，因此不建议在生产环境中使用本地盘。

1. 为 TiKV 创建附带本地存储的节点池。

```
gcloud container node-pools create tikv --cluster tidb --machine-type
    ↪ n2-highmem-8 --num-nodes=1 --local-ssd-count 1 \
  --node-labels dedicated=tikv --node-taints dedicated=tikv:NoSchedule
```

若命名为 tikv 的节点池已存在，可先删除再创建，或者修改名字规避名字冲突。

2. 部署 local volume provisioner。

   本地存储需要使用 local-volume-provisioner 程序发现并管理。以下命令会部署并创建一个 `local-storage` 的 StorageClass。

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/manifests/gke/local-ssd-provision/local-ssd-
    ↪ provision.yaml
```

3. 使用本地存储。

   完成前面步骤后，local-volume-provisioner 即可发现集群内所有本地 SSD 盘。在 `tidb-cluster.yaml` 中添加 `tikv.storageClassName` 字段并设置为 `local-storage` 即可。

4.1.2.6 部署 TiDB Operator

参考快速上手中部署 TiDB Operator，在 GKE 集群中部署 TiDB Operator。

4.1.2.7 部署 TiDB 集群和监控

下面介绍如何在 GCP GKE 上部署 TiDB 集群和监控组件。

4.1.2.7.1 创建 namespace

执行以下命令，创建 TiDB 集群安装的 namespace：

```
kubectl create namespace tidb-cluster
```

> 注意：
>
> 这里创建的 namespace 是指 Kubernetes 命名空间。本文档使用 `tidb-cluster` 为例，若使用了其他名字，修改相应的 `-n` 或 `--namespace` 参数为对应的名字即可。

4.1.2.7.2 部署 TiDB 集群

首先执行以下命令，下载 TidbCluster 和 TidbMonitor CR 的配置文件。

```
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ examples/gcp/tidb-cluster.yaml &&
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ examples/gcp/tidb-monitor.yaml
```

如需了解更详细的配置信息或者进行自定义配置，请参考配置 TiDB 集群

执行以下命令，在 GKE 集群中部署 TidbCluster 和 TidbMonitor CR。

```
kubectl create -f tidb-cluster.yaml -n tidb-cluster && \
kubectl create -f tidb-monitor.yaml -n tidb-cluster
```

当上述 yaml 文件被应用到 Kubernetes 集群后，TiDB Operator 会负责根据 yaml 文件描述，创建对应配置的 TiDB 集群。

> 注意：
>
> 如果要将 TiDB 集群部署到 ARM64 机器上，可以参考在 ARM64 机器上部署 TiDB 集群。

4.1.2.7.3 查看 TiDB 集群启动状态

使用以下命令查看 TiDB 集群启动状态：

```
kubectl get pods -n tidb-cluster
```

当所有 pods 都处于 Running & Ready 状态时，则可以认为 TiDB 集群已经成功启动。如下是一个正常运行的 TiDB 集群的示例输出：

```
NAME                              READY   STATUS    RESTARTS   AGE
tidb-discovery-5cb8474d89-n8cxk   1/1     Running   0          47h
tidb-monitor-6fbcc68669-dsjlc     3/3     Running   0          47h
tidb-pd-0                         1/1     Running   0          47h
tidb-pd-1                         1/1     Running   0          46h
tidb-pd-2                         1/1     Running   0          46h
tidb-tidb-0                       2/2     Running   0          47h
tidb-tidb-1                       2/2     Running   0          46h
tidb-tikv-0                       1/1     Running   0          47h
tidb-tikv-1                       1/1     Running   0          47h
tidb-tikv-2                       1/1     Running   0          47h
```

4.1.2.8 访问数据库

4.1.2.8.1 准备一台堡垒机

我们为 TiDB 集群创建的是内网 LoadBalancer。我们可在集群 VPC 内创建一台堡垒机来访问数据库。

```
gcloud compute instances create bastion \
    --machine-type=n1-standard-4 \
    --image-project=centos-cloud \
    --image-family=centos-7 \
    --zone=${your-region}-a
```

注意：

`${your-region}-a` 为集群所在的区域的 a 可用区，比如 us-central1-a。也可在同区域下的其他可用区创建堡垒机。

4.1.2.8.2 安装 MySQL 客户端并连接

待创建好堡垒机后，我们可以通过 SSH 远程连接到堡垒机，再通过 MySQL 客户端来访问 TiDB 集群。

1. 用 SSH 连接到堡垒机：

```
gcloud compute ssh tidb@bastion
```

2. 安装 MySQL 客户端：

```
sudo yum install mysql -y
```

3. 连接到 TiDB 集群：

```
mysql -h ${tidb-nlb-dnsname} -P 4000 -u root
```

`${tidb-nlb-dnsname}` 为 TiDB Service 的 LoadBalancer IP，可以通过 `kubectl get svc basic-tidb -n tidb-cluster` 输出中的 `EXTERNAL-IP` 字段查看。

示例：

```
$ mysql -h 10.128.15.243 -P 4000 -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 7823
Server version: 5.7.25-TiDB-v5.0.6 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 5.7 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)]> show status;
+--------------------+--------------------------------------+
| Variable_name      | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher         |                                      |
| Ssl_cipher_list    |                                      |
| Ssl_verify_mode    | 0                                    |
| Ssl_version        |                                      |
| ddl_schema_version | 22                                   |
| server_id          | 717420dc-0eeb-4d4a-951d-0d393aff295a |
+--------------------+--------------------------------------+
6 rows in set (0.01 sec)
```

注意：

- MySQL 8.0 默认认证插件从 `mysql_native_password` 更新为 `caching_sha2_password`，因此如果使用 MySQL 8.0 客户端访问 TiDB 服务（TiDB 版本 < v4.0.7），并且用户账户有配置密码，需要显式指定 `--default-auth=mysql_native_password` 参数。
- TiDB（v4.0.2 起）默认会定期收集使用情况信息，并将这些信息分享给 PingCAP 用于改善产品。若要了解所收集的信息详情及如何禁用该行为，请参见遥测。

4.1.2.8.3 访问 Grafana 监控

先获取 Grafana 的 LoadBalancer 域名：

```
kubectl -n tidb-cluster get svc basic-grafana
```

示例：

```
$ kubectl -n tidb-cluster get svc basic-grafana
NAME                  TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)
    ↪             AGE
basic-grafana         LoadBalancer 10.15.255.169 34.123.168.114 3000:30657/
    ↪ TCP      35m
```

其中 `EXTERNAL-IP` 栏即为 LoadBalancer IP。

你可以通过浏览器访问 `${grafana-lb}:3000` 地址查看 Grafana 监控指标。其中 `${grafana-lb}` 替换成前面获取的 IP。

注意：

Grafana 默认用户名和密码均为 admin。

4.1.2.9 升级 TiDB 集群

要升级 TiDB 集群，可以通过 `kubectl edit tc basic -n tidb-cluster` 命令修改 `spec.version`。

升级过程会持续一段时间，你可以通过 `kubectl get pods -n tidb-cluster --watch` 命令持续观察升级进度。

4.1.2.10 扩容 TiDB 集群

扩容前需要对相应的节点组进行扩容，以便新的实例有足够的资源运行。以下展示扩容 GKE 节点组和 TiDB 集群组件的操作。

4.1.2.10.1 扩容 GKE 节点组

下面是将 GKE 集群 `tidb` 的 `tikv` 节点池扩容到 6 节点的示例：

```
gcloud container clusters resize tidb --node-pool tikv --num-nodes 2
```

注意：

在区域集群下，节点分别创建在 3 个可用区下。这里扩容后，节点数为 $2 * 3 = 6$ 个。

4.1.2.10.2 扩容 TiDB 组件

然后通过 `kubectl edit tc basic -n tidb-cluster` 修改各组件的 `replicas` 为期望的新副本数进行扩容。

更多节点池管理可参考 Node Pools 文档。

4.1.2.11 部署 TiFlash/TiCDC

TiFlash 是 TiKV 的列存扩展，TiCDC 是一款通过拉取 TiKV 变更日志实现的 TiDB 增量数据同步工具。这两个组件不是必选安装项，这里提供一个快速安装上手示例。

4.1.2.11.1 新增节点组

为 TiFlash 新增节点组：

```
gcloud container node-pools create tiflash --cluster tidb --machine-type n1-
    ↪ highmem-8 --num-nodes=1 \
    --node-labels dedicated=tiflash --node-taints dedicated=tiflash:
        ↪ NoSchedule
```

为 TiCDC 新增节点组：

```
gcloud container node-pools create ticdc --cluster tidb --machine-type n1-
    ↪ standard-4 --num-nodes=1 \
    --node-labels dedicated=ticdc --node-taints dedicated=ticdc:NoSchedule
```

4.1.2.11.2 配置并部署 TiFlash/TiCDC

如果要部署 TiFlash，可以在 tidb-cluster.yaml 中配置 `spec.tiflash`，例如：

```
spec:
  ...
  tiflash:
    baseImage: pingcap/tiflash
    replicas: 1
    storageClaims:
    - resources:
        requests:
          storage: 100Gi
    nodeSelector:
      dedicated: tiflash
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: tiflash
```

其他参数可以参考TiDB 集群配置文档进行配置。

> **警告：**
>
> 由于 TiDB Operator 会按照 `storageClaims` 列表中的配置按顺序自动挂载 PV，如果需要为 TiFlash 增加磁盘，请确保只在列表原有配置末尾添加，并且不能修改列表中原有配置的顺序。

如果要部署 TiCDC，可以在 tidb-cluster.yaml 中配置 `spec.ticdc`，例如：

```
spec:
  ...
  ticdc:
    baseImage: pingcap/ticdc
    replicas: 1
    nodeSelector:
      dedicated: ticdc
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: ticdc
```

根据实际情况修改 `replicas` 等参数。

最后使用 `kubectl -n tidb-cluster apply -f tidb-cluster.yaml` 更新 TiDB 集群配置。

更多可参考 API 文档和集群配置文档完成 CR 文件配置。

4.1.3 在 Azure AKS 上部署 TiDB 集群

本文介绍了如何在 Azure AKS (Azure Kubernetes Service) 上部署 TiDB 集群。

如果需要部署 TiDB Operator 及 TiDB 集群到自托管 Kubernetes 环境，请参考部署 TiDB Operator及部署 TiDB 集群等文档。

4.1.3.1 前提条件

- 已安装 Helm 3，用于安装 TiDB Operator。
- 已根据部署 Azure Kubernetes 服务 (AKS) 群集 安装并配置 AKS 的命令行工具 az cli。

> **注意：**
> 可运行 `az login` 命令验证 AZ CLI 的配置是否正确。如果登陆账户成功，则 AZ CLI 的配置是正确的。否则，您需要重新配置 AZ CLI。

- 已根据使用 Azure Kubernetes 服务上的 Azure 超级磁盘（预览） 创建可以使用超级磁盘的新集群或启用现有集群上的超级磁盘。
- 已获取 AKS 服务权限。

- 在 Kubernetes 版本 < 1.21 的集群中已安装 **aks-preview CLI 扩展以使用超级磁盘，并在您的订阅中注册过 EnableAzureDiskFileCSIDriver 功能。**

  执行以下命令，安装 aks-preview CLI 扩展：

```
az extension add --name aks-preview
```

  执行以下命令，在您的 Azure 订阅中注册 EnableAzureDiskFileCSIDriver 功能：

```
az feature register --name EnableAzureDiskFileCSIDriver --namespace
    ↪ Microsoft.ContainerService --subscription ${your-subscription-id}
```

4.1.3.2 创建 AKS 集群和节点池

TiDB 集群大部分组件使用 Azure 磁盘作为存储，根据 AKS 中的最佳做法 ，推荐在创建 AKS 集群的时候确保每个节点池使用一个可用区（至少 3 个可用区）。

4.1.3.2.1 创建 启用容器存储接口 (CSI) 驱动程序 的 AKS 集群

> 注意：
>
> 在 Kubernetes 版本 < 1.21 的集群中，需要额外使用 `--aks-custom-headers` 标志来启用 **EnableAzureDiskFileCSIDriver** 特性

```
az aks create \
    --resource-group ${resourceGroup} \
    --name ${clusterName} \
    --location ${location} \
    --generate-ssh-keys \
    --vm-set-type VirtualMachineScaleSets \
    --load-balancer-sku standard \
    --node-count 3 \
    --zones 1 2 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true
```

4.1.3.2.2 创建组件节点池

集群创建成功后，执行如下命令创建组件节点池，每个节点池创建耗时约 2~5 分钟。可以参考`az aks` 文档 和 `az aks nodepool` 文档 了解更多集群配置选项。推荐在 TiKV 组件节点池启用超级磁盘。

1. 创建 operator & monitor 节点池：

```
az aks nodepool add --name admin \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --zones 1 2 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 1 \
    --labels dedicated=admin
```

2. **创建** pd **节点池**, nodeType **建议为** Standard_F4s_v2 **或更高配置：**

```
az aks nodepool add --name pd \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --node-vm-size ${nodeType} \
    --zones 1 2 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 3 \
    --labels dedicated=pd \
    --node-taints dedicated=pd:NoSchedule
```

3. **创建** tidb **节点池**, nodeType **建议为** Standard_F8s_v2 **或更高配置，默认只需要两个** TiDB **节点，因此可以设置** --node-count **为** 2**，支持修改该参数进行扩容：**

```
az aks nodepool add --name tidb \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --node-vm-size ${nodeType} \
    --zones 1 2 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 2 \
    --labels dedicated=tidb \
    --node-taints dedicated=tidb:NoSchedule
```

4. **创建** tikv **节点池**, nodeType **建议为** Standard_E8s_v4 **或更高配置：**

```
az aks nodepool add --name tikv \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --node-vm-size ${nodeType} \
    --zones 1 2 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 3 \
    --labels dedicated=tikv \
    --node-taints dedicated=tikv:NoSchedule \
    --enable-ultra-ssd
```

4.1.3.2.3 在可用区部署节点池

Azure AKS 集群使用 “尽量实现区域均衡” 在多个可用区间部署节点，如果您希望使用 “严格执行区域均衡” (AKS 暂时不支持该策略)，可以考虑在每一个可用区部署一个节点池。例如：

1. 在可用区 1 创建 tikv 节点池 1：

```
az aks nodepool add --name tikv1 \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --node-vm-size ${nodeType} \
    --zones 1 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 1 \
    --labels dedicated=tikv \
    --node-taints dedicated=tikv:NoSchedule \
    --enable-ultra-ssd
```

2. 在可用区 2 创建 tikv 节点池 2：

```
az aks nodepool add --name tikv2 \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --node-vm-size ${nodeType} \
    --zones 2 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 1 \
    --labels dedicated=tikv \
    --node-taints dedicated=tikv:NoSchedule \
    --enable-ultra-ssd
```

3. 在可用区 3 创建 tikv 节点池 3：

```
az aks nodepool add --name tikv3 \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --node-vm-size ${nodeType} \
    --zones 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 1 \
    --labels dedicated=tikv \
    --node-taints dedicated=tikv:NoSchedule \
    --enable-ultra-ssd
```

警告：

关于节点池扩缩容：

- 如果应用程序需要更改资源，可以手动缩放 AKS 群集以运行不同数量的节点。节点数减少时，节点会被优雅地清空，尽量避免对正在运行的应用程序造成中断。参考在 AKS 中缩放节点数。

4.1.3.3 配置 StorageClass

为了提高存储的 IO 写入性能，推荐设置 StorageClass 的 `mountOptions` 字段，来设置存储挂载选项 `nodelalloc` 和 `noatime`。详情可见 TiDB 环境与系统配置检查

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
### ...
mountOptions:
- nodelalloc,noatime
```

4.1.3.4 部署 TiDB Operator

参考快速上手中部署 TiDB Operator，在 AKS 集群中部署 TiDB Operator。

4.1.3.5 部署 TiDB 集群和监控

下面介绍如何在 Azure AKS 上部署 TiDB 集群和监控组件。

4.1.3.5.1 创建 namespace

执行以下命令，创建 TiDB 集群安装的 namespace：

```
kubectl create namespace tidb-cluster
```

注意：

这里创建的 namespace 是指 Kubernetes 命名空间。本文档使用 `tidb-cluster` 为例，若使用了其他名字，修改相应的 `-n` 或 `--namespace` 参数为对应的名字即可。

4.1.3.5.2 部署 TiDB 集群和监控

首先执行以下命令，下载 TidbCluster 和 TidbMonitor CR 的配置文件。

```
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ examples/aks/tidb-cluster.yaml && \
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ examples/aks/tidb-monitor.yaml
```

如需了解更详细的配置信息或者进行自定义配置，请参考配置 TiDB 集群

注意：

默认情况下，`tidb-cluster.yaml` 文件中 TiDB 服务的 LoadBalancer 配置为 “internal”。这意味着 LoadBalancer 只能在集群虚拟网络内部访问，而不能在外部访问。要通过 MySQL 协议访问 TiDB，您需要使用一个堡垒机进入集群节点或使用 `kubectl port-forward`。如果您想在互联网上公开访问 TiDB，并且知晓这样做的风险，您可以在 `tidb-cluster.yaml` 文件中将以下注释删除：

```
annotations:
  service.beta.kubernetes.io/azure-load-balancer-internal: "
    ↪ true"
```

删除后，重新创建的 LoadBalancer 及其关联的 TiDB 服务将能够在外部访问。

执行以下命令，在 AKS 集群中部署 TidbCluster 和 TidbMonitor CR。

```
kubectl apply -f tidb-cluster.yaml -n tidb-cluster && \
kubectl apply -f tidb-monitor.yaml -n tidb-cluster
```

当上述 yaml 文件被应用到 Kubernetes 集群后，TiDB Operator 会负责根据 yaml 文件描述，创建对应配置的 TiDB 集群及其监控。

4.1.3.5.3 查看 TiDB 集群启动状态

使用以下命令查看 TiDB 集群启动状态：

```
kubectl get pods -n tidb-cluster
```

当所有 pods 都处于 Running & Ready 状态时，则可以认为 TiDB 集群已经成功启动。如下是一个正常运行的 TiDB 集群的示例输出：

```
NAME                              READY   STATUS    RESTARTS   AGE
tidb-discovery-5cb8474d89-n8cxk   1/1     Running   0          47h
tidb-monitor-6fbcc68669-dsjlc     3/3     Running   0          47h
tidb-pd-0                         1/1     Running   0          47h
tidb-pd-1                         1/1     Running   0          46h
tidb-pd-2                         1/1     Running   0          46h
tidb-tidb-0                       2/2     Running   0          47h
tidb-tidb-1                       2/2     Running   0          46h
tidb-tikv-0                       1/1     Running   0          47h
tidb-tikv-1                       1/1     Running   0          47h
tidb-tikv-2                       1/1     Running   0          47h
```

4.1.3.6 访问数据库

创建好 TiDB 集群后，您就可以访问数据库，进行测试和开发了。

4.1.3.6.1 访问方式

- 使用堡垒机访问数据库

我们为 TiDB 集群创建的是内网 LoadBalancer，可以通过创建堡垒机进入集群节点来访问数据库。

> 注意：
>
> 除使用堡垒机以外，也可以使用 虚拟网络对等互连 连接现有机器到集群虚拟网络。若 AKS 创建于已经存在的虚拟网络中，可使用虚拟网络内现有机器。

- 使用 SSH 访问数据库

使用创建与 Linux 节点的 SSH 连接从而进入集群节点来访问数据库。

- 使用 node-shell 访问数据库

简单的使用 node-shell 等工具进入集群节点，然后访问数据库。

4.1.3.6.2 安装 MySQL 客户端并连接

登陆集群节点后，我们可以通过 MySQL 客户端来访问 TiDB 集群。

在集群节点上安装 MySQL 客户端：

```
sudo yum install mysql -y
```

连接到 TiDB 集群：

```
mysql --comments -h ${tidb-lb-ip} -P 4000 -u root
```

其中 `${tidb-lb-ip}` 为 TiDB Service 的 LoadBalancer 域名，可以通过命令 `kubectl get svc basic-tidb -n tidb-cluster` 输出中的 `EXTERNAL-IP` 字段查看。

以下为一个连接 TiDB 集群的示例：

```
mysql --comments -h 20.240.0.7 -P 4000 -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 1189
Server version: 5.7.25-TiDB-v4.0.2 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 5.7 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)]> show status;
+--------------------+--------------------------------------+
| Variable_name      | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher         |                                      |
| Ssl_cipher_list    |                                      |
| Ssl_verify_mode    | 0                                    |
| Ssl_version        |                                      |
| ddl_schema_version | 22                                   |
| server_id          | ed4ba88b-436a-424d-9087-977e897cf5ec |
+--------------------+--------------------------------------+
6 rows in set (0.00 sec)
```

注意：

- MySQL 8.0 默认认证插件从 `mysql_native_password` 更新为 `caching_sha2_password`，因此如果使用 MySQL 8.0 客户端访问 TiDB

服务（TiDB 版本 < v4.0.7），并且用户账户有配置密码，需要显示指定 `--default-auth=mysql_native_password` 参数。
- TiDB（v4.0.2 起）默认会定期收集使用情况信息，并将这些信息分享给 PingCAP 用于改善产品。若要了解所收集的信息详情及如何禁用该行为，请参见 TiDB 遥测功能使用文档。

4.1.3.7 访问 Grafana 监控

先获取 Grafana 的 LoadBalancer 域名：

```
kubectl -n tidb-cluster get svc basic-grafana
```

示例输出：

```
kubectl get svc basic-grafana
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP
    ↪                                                      PORT(S)          AGE
basic-grafana LoadBalancer 10.100.199.42 20.240.0.8   3000:30761/TCP 121m
```

其中 `EXTERNAL-IP` 栏即为 LoadBalancer 域名。

您可以通过浏览器访问 `${grafana-lb}:3000` 地址查看 Grafana 监控指标。其中 `${grafana-lb}` 替换成前面获取的域名。

注意：

Grafana 默认用户名和密码均为 admin。

4.1.3.8 访问 TiDB Dashboard

如果想要安全地访问 TiDB Dashboard，详情可以参见访问 TiDB Dashboard。

4.1.3.9 升级 TiDB 集群

要升级 TiDB 集群，可以通过 `kubectl edit tc basic -n tidb-cluster` 命令修改 `spec.version`。

升级过程会持续一段时间，您可以通过 `kubectl get pods -n tidb-cluster --watch` 命令持续观察升级进度。

4.1.3.10 扩容 TiDB 集群

扩容前需要对相应的节点池进行扩容，以便新的实例有足够的资源运行。以下展示扩容 AKS 节点池和 TiDB 集群组件的操作。

4.1.3.10.1 扩容 AKS 节点池

TiKV 扩容需要保证在各可用区均匀扩容。以下是将集群 `${clusterName}` 的节点池扩容到 6 节点的示例：

```
az aks nodepool scale \
    --resource-group ${resourceGroup} \
    --cluster-name ${clusterName} \
    --name ${nodePoolName} \
    --node-count 6
```

更多节点池管理可参考 az aks nodepool 文档。

4.1.3.10.2 扩容 TiDB 组件

扩容 AKS 节点池后，可以使用命令 `kubectl edit tc basic -n tidb-cluster` 修改各组件的 `replicas` 为期望的新副本数进行扩容。

4.1.3.11 部署 TiFlash/TiCDC

TiFlash 是 TiKV 的列存扩展。

TiCDC 是一款通过拉取 TiKV 变更日志实现的 TiDB 增量数据同步工具。

这两个组件不是必选安装项，这里提供一个快速安装上手示例。

4.1.3.11.1 新增节点池

为 TiFlash/TiCDC 各自新增一个节点池。`--node-count` 决定期望的节点数，根据实际需求而定。

- 创建 tiflash 节点池, nodeType 建议为 Standard_E8s_v4 或更高配置：

```
az aks nodepool add --name tiflash \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
    --node-vm-size ${nodeType} \
    --zones 1 2 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 3 \
    --labels dedicated=tiflash \
    --node-taints dedicated=tiflash:NoSchedule
```

- 创建 ticdc 节点池, nodeType 建议为 Standard_E16s_v4 或更高配置：

```
az aks nodepool add --name ticdc \
    --cluster-name ${clusterName} \
    --resource-group ${resourceGroup} \
```

```
    --node-vm-size ${nodeType} \
    --zones 1 2 3 \
    --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
    --node-count 3 \
    --labels dedicated=ticdc \
    --node-taints dedicated=ticdc:NoSchedule
```

4.1.3.11.2 配置并部署 TiFlash/TiCDC

如果要部署 TiFlash，可以在 tidb-cluster.yaml 中配置 `spec.tiflash`，例如：

```
spec:
  ...
  tiflash:
    baseImage: pingcap/tiflash
    replicas: 1
    storageClaims:
    - resources:
        requests:
          storage: 100Gi
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: tiflash
```

其他参数可以参考TiDB 集群配置文档进行配置。

> **警告：**
>
> 由于 TiDB Operator 会按照 `storageClaims` 列表中的配置按顺序自动挂载 PV，如果需要为 TiFlash 增加磁盘，请确保只在列表原有配置末尾添加，并且不能修改列表中原有配置的顺序。

如果要部署 TiCDC，可以在 tidb-cluster.yaml 中配置 `spec.ticdc`，例如：

```
spec:
  ...
  ticdc:
    baseImage: pingcap/ticdc
    replicas: 1
    tolerations:
    - effect: NoSchedule
```

```
    key: dedicated
    operator: Equal
    value: ticdc
```

根据实际情况修改 replicas 等参数。

最后使用 kubectl -n tidb-cluster apply -f tidb-cluster.yaml 更新 TiDB 集群配置。

更多可参考 API 文档和集群配置文档完成 CR 文件配置。

4.1.3.12 使用其他 Azure 磁盘类型

Azure Disk 支持多种磁盘类型。若需要低延迟、高吞吐，可以选择 UltraSSD 类型。首先我们为 UltraSSD 新建一个存储类 (Storage Class)：

1. 启用现有群集上的超级磁盘 并创建存储类 ultra:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ultra
provisioner: disk.csi.azure.com
parameters:
  skuname: UltraSSD_LRS # alias: storageaccounttype, available values:
      ↪ Standard_LRS, Premium_LRS, StandardSSD_LRS, UltraSSD_LRS
  cachingMode: None
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
mountOptions:
- nodelalloc,noatime
```

   你可以根据实际需要额外配置驱动参数。

2. 然后在 tidb cluster 的 YAML 文件中，通过 storageClassName 字段指定 ultra 存储类申请 UltraSSD 类型的 Azure 磁盘。可以参考以下 TiKV 配置示例使用：

```
spec:
  tikv:
    baseImage: pingcap/tikv
    replicas: 3
    storageClassName: ultra
    requests:
      storage: "100Gi"
```

您可以使用任意 Azure 磁盘类型，推荐使用 `Premium_LRS` 或 `UltraSSD_LRS`。

更多关于存储类配置和 Azure 磁盘类型的信息，可以参考 Storage Class 官方文档和 Azure 磁盘类型官方文档。

4.1.3.13 使用本地存储

请使用 Azure LRS Disk 作为生产环境的存储类型。如果需要模拟测试裸机部署的性能，可以使用 Azure 部分实例类型提供的 NVMe SSD 本地磁盘。可以为 TiKV 节点池选择这一类型的实例，以便提供更高的 IOPS 和低延迟。

> **注意：**
>
> 运行中的 TiDB 集群不能动态更换 storage class，可创建一个新的 TiDB 集群测试。
>
> 本地 NVMe 磁盘是临时的，如果停止/解除分配 VM，这些磁盘上的数据都将丢失。由于 AKS 升级或其他原因造成的节点重建，会导致需要迁移 TiKV 数据，如果无法接受这一点，则不建议在生产环境中使用本地磁盘。

了解哪些实例可提供本地磁盘，可以查看 Lsv2 系列。以下以 `Standard_L8s_v2` 为例：

1. 为 TiKV 创建附带本地磁盘的节点池。

   修改 `az aks nodepool add` 命令中 TiKV 节点池实例类型为 `Standard_L8s_v2`：

   ```
   az aks nodepool add --name tikv \
       --cluster-name ${clusterName} \
       --resource-group ${resourceGroup} \
       --node-vm-size Standard_L8s_v2 \
       --zones 1 2 3 \
       --aks-custom-headers EnableAzureDiskFileCSIDriver=true \
       --node-count 3 \
       --enable-ultra-ssd \
       --labels dedicated=tikv \
       --node-taints dedicated=tikv:NoSchedule
   ```

   若 tikv 节点池已存在，可先删除再创建，或者修改名字规避冲突。

2. 部署 local volume provisioner。

   本地存储需要使用 local-volume-provisioner 程序发现并管理。以下命令会部署并创建一个 `local-storage` 的 Storage Class。

   ```
   kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
       ↪ operator/v1.1.15/manifests/eks/local-volume-provisioner.yaml
   ```

3. 使用本地存储。

   完成前面步骤后，local-volume-provisioner 即可发现集群内所有本地 NVMe SSD 盘。在 tidb-cluster.yaml 中添加 `tikv.storageClassName` 字段并设置为 `local-storage` 即可，可以参考前文部署 TiDB 集群和监控部分。

4.1.4 在阿里云上部署 TiDB 集群

本文介绍了如何使用个人电脑（Linux 或 macOS 系统）在阿里云上部署 TiDB 集群。

如果需要部署 TiDB Operator 及 TiDB 集群到自托管 Kubernetes 环境，请参考部署 TiDB Operator及部署 TiDB 集群等文档。

4.1.4.1 环境需求

- aliyun-cli >= 3.0.15 并且配置 aliyun-cli

> **注意：**
> Access Key 需要具有操作相应资源的权限。

- kubectl >= 1.12
- Helm 3
- jq >= 1.6
- terraform 0.12.*

你可以使用阿里云的云命令行服务来进行操作，云命令行中已经预装并配置好了所有工具。

4.1.4.1.1 权限

完整部署集群需要具备以下权限：

- AliyunECSFullAccess
- AliyunESSFullAccess
- AliyunVPCFullAccess
- AliyunSLBFullAccess
- AliyunCSFullAccess
- AliyunEIPFullAccess
- AliyunECIFullAccess
- AliyunVPNGatewayFullAccess
- AliyunNATGatewayFullAccess

4.1.4.2 概览

默认配置下，会创建：

- 一个新的 VPC
- 一台 ECS 实例作为堡垒机
- 一个托管版 ACK（阿里云 Kubernetes）集群以及一系列 worker 节点：
  - 属于一个伸缩组的 2 台 ECS 实例（2 核 2 GB），托管版 Kubernetes 的默认伸缩组中必须至少有两台实例，用于承载整个的系统服务，例如 CoreDNS
  - 属于一个伸缩组的 3 台 `ecs.g5.large` 实例，用于部署 PD
  - 属于一个伸缩组的 3 台 `ecs.i2.2xlarge` 实例，用于部署 TiKV
  - 属于一个伸缩组的 2 台 `ecs.c5.4xlarge` 实例用于部署 TiDB
  - 属于一个伸缩组的 1 台 `ecs.c5.xlarge` 实例用于部署监控组件

除了默认伸缩组之外的其它所有实例都是跨可用区部署的。而伸缩组 (Auto-scaling Group) 能够保证集群的健康实例数等于期望数值。因此，当发生节点故障甚至可用区故障时，伸缩组能够自动为我们创建新实例来确保服务可用性。

4.1.4.3 安装部署

4.1.4.3.1 部署 ACK，TiDB Operator 和 TiDB 集群节点池

使用如下步骤部署 ACK，TiDB Operator 和 TiDB 集群节点池。

1. 设置目标 region 和阿里云密钥（也可以在运行 `terraform` 命令时根据命令提示输入）：

```
export TF_VAR_ALICLOUD_REGION=${REGION} && \
export TF_VAR_ALICLOUD_ACCESS_KEY=${ACCESS_KEY} && \
export TF_VAR_ALICLOUD_SECRET_KEY=${SECRET_KEY}
```

用于部署集群的各变量的默认值存储在 `variables.tf` 文件中，如需定制可以修改此文件或在安装时通过 `-var` 参数覆盖。

2. 使用 Terraform 进行安装：

```
git clone --depth=1 https://github.com/pingcap/tidb-operator && \
cd tidb-operator/deploy/aliyun
```

可以新建或者编辑 `terraform.tfvars`，在其中设置变量的值，按需配置集群，可以通过 `variables.tf` 查看有哪些变量可以设置以及各变量的详细描述。例如，下面示例配置 ACK 集群名称、TiDB 集群名称，TiDB Operator 版本及 PD、TiKV 和 TiDB 节点的数量：

```
cluster_name = "testack"
tidb_cluster_name = "testdb"
tikv_count = 3
tidb_count = 2
pd_count = 3
operator_version = "v1.1.15"
```

如果需要在集群中部署 TiFlash，需要在 terraform.tfvars 中设置 create_tiflash_node_pool = true，也可以设置 tiflash_count 和 tiflash_instance_type 来配置 TiFlash 节点池的节点数量和实例类型，tiflash_count 默认为 2，tiflash_instance_type 默认为 ecs.i2.2xlarge。

如果需要在集群中部署 TiCDC，需要在 terraform.tfvars 中设置 create_cdc_node_pool = true，也可以设置 cdc_count 和 cdc_instance_type 来配置 TiCDC 节点池的节点数量和实例类型，cdc_count 默认为 3，cdc_instance_type 默认为 ecs.c5.2xlarge。

> 注意：
> 请通过 variables.tf 文件中的 operator_version 确认当前版本脚本中默认的 TiDB Operator 版本，如果默认版本不是想要使用的版本，请在 terraform.tfvars 中配置 operator_version。

配置完成后，使用 terraform 命令初始化并部署集群：

```
terraform init
```

apply 过程中需要输入 yes 来确认执行：

```
terraform apply
```

假如在运行 terraform apply 时出现报错，可根据报错信息（例如缺少权限）进行修复后再次运行 terraform apply。

整个安装过程大约需要 5 至 10 分钟，安装完成后会输出集群的关键信息（想要重新查看这些信息，可以运行 terraform output）：

```
Apply complete! Resources: 3 added, 0 changed, 1 destroyed.

Outputs:

bastion_ip = 47.96.174.214
cluster_id = c2d9b20854a194f158ef2bc8ea946f20e
kubeconfig_file = /tidb-operator/deploy/aliyun/credentials/kubeconfig
monitor_endpoint = not_created
region = cn-hangzhou
```

```
ssh_key_file = /tidb-operator/deploy/aliyun/credentials/my-cluster-keyZ
    ↪ .pem
tidb_endpoint = not_created
tidb_version = v3.0.0
vpc_id = vpc-bp1v8i5rwsc7yh8dwyep5
```

3. 用 kubectl 或 helm 对集群进行操作：

```
export KUBECONFIG=$PWD/credentials/kubeconfig
```

```
kubectl version
```

```
helm ls
```

4.1.4.3.2 部署 TiDB 集群和监控

1. 准备 TidbCluster 和 TidbMonitor CR 文件：

```
cp manifests/db.yaml.example db.yaml && cp manifests/db-monitor.yaml.
    ↪ example db-monitor.yaml
```

参考 API 文档和集群配置文档完成 CR 文件配置。

如果要部署 TiFlash，可以在 db.yaml 中配置 spec.tiflash，例如：

```
spec
  ...
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 3
    nodeSelector:
      dedicated: TIDB_CLUSTER_NAME-tiflash
    replicas: 1
    storageClaims:
    - resources:
        requests:
          storage: 100Gi
      storageClassName: local-volume
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: TIDB_CLUSTER_NAME-tiflash
```

其他参数可以参考集群配置文档进行配置。

根据实际情况修改 replicas、storageClaims[].resources.requests.storage、storageClassName。

> **警告：**
>
> 由于 TiDB Operator 会按照 storageClaims 列表中的配置按顺序自动挂载 PV，如果需要为 TiFlash 增加磁盘，请确保只在列表原有配置最后添加，并且不能修改列表中原有配置的顺序。

如果要部署 TiCDC，可以在 db.yaml 中配置 spec.ticdc，例如：

```
spec
  ...
  ticdc:
    baseImage: pingcap/ticdc
    nodeSelector:
      dedicated: TIDB_CLUSTER_NAME-cdc
    replicas: 3
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: TIDB_CLUSTER_NAME-cdc
```

根据实际情况修改 replicas。

> **注意：**
>
> - 请使用 ACK 部署过程中配置的 tidb_cluster_name 替换 db.yaml 和 db-monitor.yaml 文件中所有的 TIDB_CLUSTER_NAME。
> - 请确保 ACK 部署过程中 PD、TiKV、TiFlash、TiCDC 或者 TiDB 节点的数量的值大于等于 db.yaml 中对应组件的 replicas。
> - 请确保 db-monitor.yaml 中 spec.initializer.version 和 db.yaml 中 spec.version 一致，以保证监控显示正常。

2. 创建 Namespace：

```
kubectl --kubeconfig credentials/kubeconfig create namespace ${
    ↪ namespace}
```

> **注意：**
>
> namespace 是命名空间，可以起一个方便记忆的名字，比如和 tidb_cluster_name 相同的名称。

3. 部署 TiDB 集群：

```
shell  kubectl --kubeconfig credentials/kubeconfig create -f db.yaml -n
↪ ${namespace} && kubectl --kubeconfig credentials/kubeconfig create -f db
↪ -monitor.yaml -n ${namespace}
```

> **注意：**
>
> 如果要将 TiDB 集群部署到 ARM64 机器上，可以参考在 ARM64 机器上部署 TiDB 集群。

4.1.4.4 连接数据库

通过堡垒机可连接 TiDB 集群进行测试，相关信息在安装完成后的输出中均可找到：

```
ssh -i credentials/${cluster_name}-key.pem root@${bastion_ip}
```

```
mysql -h ${tidb_lb_ip} -P 4000 -u root
```

tidb_lb_ip 为 TiDB Service 的 LoadBalancer IP。

> **注意：**
>
> - MySQL 8.0 默认认证插件从 mysql_native_password 更新为 caching_sha2_password，因此如果使用 MySQL 8.0 客户端访问 TiDB 服务（TiDB 版本 < v4.0.7），并且用户账户有配置密码，需要显示指定 --default-auth=mysql_native_password 参数。
> - TiDB（v4.0.2 起）默认会定期收集使用情况信息，并将这些信息分享给 PingCAP 用于改善产品。若要了解所收集的信息详情及如何禁用该行为，请参见遥测。

4.1.4.5 监控

你可以通过浏览器访问 <monitor-lb>:3000 地址查看 Grafana 监控指标。

monitor-lb 是集群 Monitor Service 的 LoadBalancer IP。

默认帐号密码为：

- 用户名：admin
- 密码：admin

> 警告：
>
> 出于安全考虑，假如你已经或将要配置 VPN 用于访问 VPC，强烈建议将 db-monitor.yaml 文件里 spec.grafana.service.annotations 中的 service.beta.kubernetes.io/alicloud-loadbalancer-address-type 设置为 intranet 以禁止监控服务的公网访问。

4.1.4.6 升级 TiDB 集群

要升级 TiDB 集群，可以通过 `kubectl --kubeconfig credentials/kubeconfig edit tc ${tidb_cluster_name} -n ${namespace}` 修改 spec.version。

升级操作可能会执行较长时间，可以通过以下命令来持续观察进度：

```
kubectl get pods --namespace ${namespace} -o wide --watch
```

4.1.4.7 TiDB 集群扩容

若要扩容 TiDB 集群，可以在文件 terraform.tfvars 文件中设置 tikv_count、tiflash_count、cdc_count 或者 tidb_count 变量，然后运行 terraform apply，扩容对应组件节点数量，节点扩容完成后，通过 `kubectl --kubeconfig credentials/kubeconfig edit tc ${tidb_cluster_name} -n ${namespace}` 修改对应组件的 replicas。

> 注意：
>
> - 由于缩容过程中无法确定会缩掉哪个节点，目前还不支持 TiDB 集群的缩容。
> - 扩容过程会持续几分钟，你可以通过 `kubectl --kubeconfig credentials/kubeconfig get po -n ${namespace} --watch` 命令持续观察进度。

4.1.4.8 销毁集群

可以参考销毁 TiDB 集群删除集群。

然后通过如下命令销毁 ACK 集群：

```
terraform destroy
```

假如 Kubernetes 集群没有创建成功，那么在 destroy 时会出现报错，无法进行正常清理。此时需要手动将 Kubernetes 资源从本地状态中移除：

```
terraform state list
```

```
terraform state rm module.ack.alicloud_cs_managed_kubernetes.k8s
```

销毁集群操作需要执行较长时间。

> 注意：
> 组件挂载的云盘需要在阿里云管理控制台中手动删除。

4.1.4.9 配置

4.1.4.9.1 配置 TiDB Operator

通过在 `terraform.tfvars` 中设置变量的值来配置 TiDB Operator，大多数配置项均能按照 `variable` 的注释理解语义后进行修改。需要注意的是，`operator_helm_values` 配置项允许为 TiDB Operator 提供一个自定义的 `values.yaml` 配置文件，示例如下：

- 在 `terraform.tfvars` 中设置 `operator_helm_values`：

  ```
  operator_helm_values = "./my-operator-values.yaml"
  ```

- 在 `main.tf` 中设置 `operator_helm_values`：

  ```
  operator_helm_values = file("./my-operator-values.yaml")
  ```

同时，在默认配置下 Terraform 脚本会创建一个新的 VPC，假如要使用现有的 VPC，可以在 `variable.tf` 中设置 `vpc_id`。注意，当使用现有 VPC 时，没有设置 vswitch 的可用区将不会部署 Kubernetes 节点。

4.1.4.9.2 配置 TiDB 集群

参考 API 文档和集群配置文档修改 TiDB 集群配置。

4.1.4.10 管理多个 TiDB 集群

需要在一个 Kubernetes 集群下管理多个 TiDB 集群时，需要编辑 `./main.tf`，按实际需要新增 `tidb-cluster` 声明，示例如下：

```
module "tidb-cluster-dev" {
  source = "../modules/aliyun/tidb-cluster"
  providers = {
    helm = helm.default
  }

  cluster_name = "dev-cluster"
  ack         = module.tidb-operator

  pd_count                = 1
  tikv_count              = 1
  tidb_count              = 1
}

module "tidb-cluster-staging" {
  source = "../modules/aliyun/tidb-cluster"
  providers = {
    helm = helm.default
  }

  cluster_name = "staging-cluster"
  ack         = module.tidb-operator

  pd_count                = 3
  tikv_count              = 3
  tidb_count              = 2
}
```

注意，多个 TiDB 集群之间 `cluster_name` 必须保持唯一。下面是 `tidb-cluster` 模块的所有可配置参数：

参数名	说明
`ack`	封装目标 Kubernetes 集群信息的结构体，必填
`cluster_name`	TiDB 集群名，必填且必须唯一
`tidb_version`	TiDB 集群版本
`tidb_cluster_chart_version`	`tidb-cluster` helm chart 的版本
`pd_count`	PD 节点数
`pd_instance_type`	PD 实例类型
`tikv_count`	TiKV 节点数
`tikv_instance_type`	TiKV 实例类型
`tiflash_count`	TiFlash 节点数

参数名	说明
tiflash_instance_type	TiFlash 实例类型
cdc_count	TiCDC 节点数
cdc_instance_type	TiCDC 实例类型
tidb_count	TiDB 节点数
tidb_instance_type	TiDB 实例类型
monitor_instance_type	监控组件的实例类型
override_values	TiDB 集群的 values.yaml 配置文件，通常通过 file() 函数从文件中
local_exec_interpreter	执行命令行指令的解释器
create_tidb_cluster_release	是否通过 Helm 创建 TiDB 集群

4.1.4.11 管理多个 Kubernetes 集群

推荐针对每个 Kubernetes 集群都使用单独的 Terraform 模块进行管理（一个 Terraform Module 即一个包含 .tf 脚本的目录）。

deploy/aliyun 实际上是将 deploy/modules 中的数个可复用的 Terraform 脚本组合在了一起。当管理多个集群时（下面的操作在 tidb-operator 项目根目录下进行）：

1. 首先针对每个集群创建一个目录，如：

```
mkdir -p deploy/aliyun-staging
```

2. 参考 deploy/aliyun 的 main.tf，编写自己的脚本，下面是一个简单的例子：

```
provider "alicloud" {
    region     = ${REGION}
    access_key = ${ACCESS_KEY}
    secret_key = ${SECRET_KEY}
}

module "tidb-operator" {
    source     = "../modules/aliyun/tidb-operator"

    region         = ${REGION}
    access_key     = ${ACCESS_KEY}
    secret_key     = ${SECRET_KEY}
    cluster_name   = "example-cluster"
    key_file       = "ssh-key.pem"
    kubeconfig_file = "kubeconfig"
}

provider "helm" {
    alias    = "default"
    insecure = true
```

```
    install_tiller = false
    kubernetes {
        config_path = module.tidb-operator.kubeconfig_filename
    }
}

module "tidb-cluster" {
    source = "../modules/aliyun/tidb-cluster"
    providers = {
        helm = helm.default
    }

    cluster_name = "example-cluster"
    ack          = module.tidb-operator
}

module "bastion" {
    source = "../modules/aliyun/bastion"

    bastion_name          = "example-bastion"
    key_name              = module.tidb-operator.key_name
    vpc_id                = module.tidb-operator.vpc_id
    vswitch_id            = module.tidb-operator.vswitch_ids[0]
    enable_ssh_to_worker  = true
    worker_security_group_id = module.tidb-operator.security_group_id
}
```

上面的脚本可以自由定制，比如，假如不需要堡垒机则可以移除 `module "bastion"` 相关声明。

你也可以直接拷贝 `deploy/aliyun` 目录，但要注意不能拷贝已经运行了 `terraform apply` 的目录，建议重新 clone 仓库再进行拷贝。

4.1.4.12 使用限制

目前，`pod cidr`，`service cidr` 和节点型号等配置在集群创建后均无法修改。

4.1.5 部署到自托管的 Kubernetes

4.1.5.1 Kubernetes 上的 TiDB 集群环境需求

本文介绍在 Kubernetes 上部署 TiDB 集群的软硬件环境需求。

4.1.5.1.1 软件版本要求

软件名称	版本
Docker	Docker CE 18.09.6
Kubernetes	v1.12.5+
CentOS	CentOS 7.6，内核要求为 3.10.0-957 或之后版本
Helm	v3.0.0+

4.1.5.1.2 配置防火墙

建议关闭防火墙：

```
systemctl stop firewalld
systemctl disable firewalld
```

如果无法关闭 firewalld 服务，为了保证 Kubernetes 正常运行，需要打开以下端口：

1. 在 Master 节点上，打开以下端口，然后重新启动服务：

```
firewall-cmd --permanent --add-port=6443/tcp
firewall-cmd --permanent --add-port=2379-2380/tcp
firewall-cmd --permanent --add-port=10250/tcp
firewall-cmd --permanent --add-port=10251/tcp
firewall-cmd --permanent --add-port=10252/tcp
firewall-cmd --permanent --add-port=10255/tcp
firewall-cmd --permanent --add-port=8472/udp
firewall-cmd --add-masquerade --permanent

# 当需要在 Master 节点上暴露 NodePort 时候设置
firewall-cmd --permanent --add-port=30000-32767/tcp

systemctl restart firewalld
```

2. 在 Node 节点上，打开以下端口，然后重新启动服务：

```
firewall-cmd --permanent --add-port=10250/tcp
firewall-cmd --permanent --add-port=10255/tcp
firewall-cmd --permanent --add-port=8472/udp
firewall-cmd --permanent --add-port=30000-32767/tcp
firewall-cmd --add-masquerade --permanent

systemctl restart firewalld
```

4.1.5.1.3 配置 Iptables

FORWARD 链默认配置成 ACCEPT，并将其设置到开机启动脚本里：

```
iptables -P FORWARD ACCEPT
```

4.1.5.1.4 禁用 SELinux

```
setenforce 0
sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

4.1.5.1.5 关闭 Swap

Kubelet 正常工作需要关闭 Swap，并且把 /etc/fstab 里面有关 Swap 的那行注释掉：

```
swapoff -a
sed -i 's/^\(.*swap.*\)$/#\1/' /etc/fstab
```

4.1.5.1.6 内核参数设置

按照下面的配置设置内核参数，也可根据自身环境进行微调：

```
modprobe br_netfilter

cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-arptables = 1
net.core.somaxconn = 32768
vm.swappiness = 0
net.ipv4.tcp_syncookies = 0
net.ipv4.ip_forward = 1
fs.file-max = 1000000
fs.inotify.max_user_watches = 1048576
fs.inotify.max_user_instances = 1024
net.ipv4.conf.all.rp_filter = 1
net.ipv4.neigh.default.gc_thresh1 = 80000
net.ipv4.neigh.default.gc_thresh2 = 90000
net.ipv4.neigh.default.gc_thresh3 = 100000
EOF

sysctl --system
```

4.1.5.1.7 配置 Irqbalance 服务

Irqbalance 服务可以将各个设备对应的中断号分别绑定到不同的 CPU 上，以防止所有中断请求都落在同一个 CPU 上而引发性能瓶颈。

```
systemctl enable irqbalance
systemctl start irqbalance
```

4.1.5.1.8 CPUfreq 调节器模式设置

为了让 CPU 发挥最大性能，请将 CPUfreq 调节器模式设置为 performance 模式。详细参考在部署目标机器上配置 CPUfreq 调节器模式。

```
cpupower frequency-set --governor performance
```

4.1.5.1.9 Ulimit 设置

TiDB 集群默认会使用很多文件描述符，需要将工作节点上面的 ulimit 设置为大于等于 1048576：

```
cat <<EOF >> /etc/security/limits.conf
root       soft       nofile      1048576
root       hard       nofile      1048576
root       soft       stack       10240
EOF

sysctl --system
```

4.1.5.1.10 Docker 服务

安装 Docker 时，建议选择 Docker CE 18.09.6 及以上版本。请参考 Docker 安装指南进行安装。

安装完 Docker 服务以后，执行以下步骤：

1. 将 Docker 的数据保存到一块单独的盘上，Docker 的数据主要包括镜像和容器日志数据。通过设置 --data-root 参数来实现：

```
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ],
  "data-root": "/data1/docker"
}
EOF
```

上面会将 Docker 的数据目录设置为 /data1/docker。

2. 设置 Docker daemon 的 ulimit。

    1. 创建 docker service 的 systemd drop-in 目录 `/etc/systemd/system/docker.service.d`：

    ```
    mkdir -p /etc/systemd/system/docker.service.d
    ```

    2. 创建 `/etc/systemd/system/docker.service.d/limit-nofile.conf` 文件，并配置 `LimitNOFILE` 参数的值，取值范围为大于等于 `1048576` 的数字即可。

    ```
    cat > /etc/systemd/system/docker.service.d/limit-nofile.conf <<EOF
    [Service]
    LimitNOFILE=1048576
    EOF
    ```

    > 注意：
    > 请勿将 `LimitNOFILE` 的值设置为 `infinity`。由于 systemd 的 bug，`infinity` 在 `systemd` 某些版本中指的是 `65536`。

    3. 重新加载配置。

    ```
    systemctl daemon-reload && systemctl restart docker
    ```

4.1.5.1.11 Kubernetes 服务

参考 Kubernetes 官方文档，部署一套多 Master 节点高可用集群。

Kubernetes Master 节点的配置取决于 Kubernetes 集群中 Node 节点个数，节点数越多，需要的资源也就越多。节点数可根据需要做微调。

Kubernetes 集群 Node 节点个数	Kubernetes Master 节点配置
1-5	1vCPUs 4GB Memory
6-10	2vCPUs 8GB Memory
11-100	4vCPUs 16GB Memory
101-250	8vCPUs 32GB Memory
251-500	16vCPUs 64GB Memory
501-5000	32vCPUs 128GB Memory

安装完 Kubelet 之后，执行以下步骤：

1. 将 Kubelet 的数据保存到一块单独盘上（可跟 Docker 共用一块盘），Kubelet 主要占盘的数据是 emptyDir 所使用的数据。通过设置 `--root-dir` 参数来实现：

```
echo "KUBELET_EXTRA_ARGS=--root-dir=/data1/kubelet" > /etc/sysconfig/
    ↪ kubelet
```

```
systemctl restart kubelet
```

上面会将 Kubelet 数据目录设置为 `/data1/kubelet`。

2. 通过 kubelet 设置预留资源，保证机器上的系统进程以及 Kubernetes 的核心进程在工作负载很高的情况下仍然有足够的资源来运行，从而保证整个系统的稳定。

4.1.5.1.12 TiDB 集群资源需求

请根据服务器建议配置来规划机器的配置。

另外，在生产环境中，尽量不要在 Kubernetes Master 节点部署 TiDB 实例，或者尽可能少地部署 TiDB 实例。因为网卡带宽的限制，Master 节点网卡满负荷工作会影响到 Worker 节点和 Master 节点之间的心跳信息汇报，导致比较严重的问题。

4.1.5.2 Kubernetes 上的持久化存储类型配置

TiDB 集群中 PD、TiKV、监控等组件以及 TiDB Binlog 和备份等工具都需要使用将数据持久化的存储。Kubernetes 上的数据持久化需要使用 PersistentVolume (PV)。Kubernetes 提供多种存储类型，主要分为两大类：

- 网络存储

  存储介质不在当前节点，而是通过网络方式挂载到当前节点。一般有多副本冗余提供高可用保证，在节点出现故障时，对应网络存储可以再挂载到其它节点继续使用。

- 本地存储

  存储介质在当前节点，通常能提供比网络存储更低的延迟，但没有多副本冗余，一旦节点出故障，数据就有可能丢失。如果是 IDC 服务器，节点故障可以一定程度上对数据进行恢复，但公有云上使用本地盘的虚拟机在节点故障后，数据是无法找回的。

PV 一般由系统管理员或 volume provisioner 自动创建，PV 与 Pod 是通过 PersistentVolumeClaim (PVC) 进行关联的。普通用户在使用 PV 时并不需要直接创建 PV，而是通过 PVC 来申请使用 PV，对应的 volume provisioner 根据 PVC 创建符合要求的 PV，并将 PVC 与该 PV 进行绑定。

> **警告：**
>
> 为了数据安全，任何情况下都不要直接删除 PV，除非对 volume provisioner 原理非常清楚。手动删除 PV 可能导致非预期的行为。

4.1.5.2.1 TiDB 集群推荐存储类型

TiKV 自身借助 Raft 实现了数据复制，出现节点故障后，PD 会自动进行数据调度补齐缺失的数据副本，同时 TiKV 要求存储有较低的读写延迟，所以生产环境强烈推荐使用本地 SSD 存储。

PD 同样借助 Raft 实现了数据复制，但作为存储集群元信息的数据库，并不是 IO 密集型应用，所以一般本地普通 SAS 盘或网络 SSD 存储 (例如 AWS 上 gp2 类型的 EBS 存储卷，GCP 上的持久化 SSD 盘) 就可以满足要求。

监控组件以及 TiDB Binlog、备份等工具，由于自身没有做多副本冗余，所以为保证可用性，推荐用网络存储。其中 TiDB Binlog 的 pump 和 drainer 组件属于 IO 密集型应用，需要较低的读写延迟，所以推荐用高性能的网络存储 (例如 AWS 上的 io1 类型的 EBS 存储卷，GCP 上的持久化 SSD 盘)。

在利用 TiDB Operator 部署 TiDB 集群或者备份工具的时候，需要持久化存储的组件都可以通过 values.yaml 配置文件中对应的 `storageClassName` 设置存储类型。不设置时默认都使用 `local-storage`。

4.1.5.2.2 网络 PV 配置

Kubernetes 1.11 及以上的版本支持网络 PV 的动态扩容，但用户需要为相应的 `StorageClass` 开启动态扩容支持。

```
kubectl patch storageclass ${storage_class} -p '{"allowVolumeExpansion":
    ↪ true}'
```

开启动态扩容后，通过下面方式对 PV 进行扩容：

1. 修改 PVC 大小

   假设之前 PVC 大小是 10 Gi，现在需要扩容到 100 Gi

   ```
   kubectl patch pvc -n ${namespace} ${pvc_name} -p '{"spec": {"resources
       ↪ ": {"requests": {"storage": "100Gi"}}}}'
   ```

2. 查看 PV 扩容成功

   扩容成功后，通过 `kubectl get pvc -n ${namespace} ${pvc_name}` 显示的大小仍然是初始大小，但查看 PV 大小会显示已经扩容到预期的大小。

   ```
   kubectl get pv | grep ${pvc_name}
   ```

4.1.5.2.3 本地 PV 配置

Kubernetes 当前支持静态分配的本地存储。可使用 local-static-provisioner 项目中的 `local-volume-provisioner` 程序创建本地存储对象。

第 1 步：准备本地存储

- 给 TiKV 数据使用的盘，可通过普通挂载方式将盘挂载到 /mnt/ssd 目录。

  出于性能考虑，推荐 TiKV 独占一个磁盘，并且推荐磁盘类型为 SSD。

- 给 PD 数据使用的盘，可以参考步骤挂载盘，创建目录，并将新建的目录以 bind mount 方式挂载到 /mnt/sharedssd 目录下。

> 注意：
> 该步骤中创建的目录个数取决于规划的 TiDB 集群数量及每个集群内的 PD 数量。1 个目录会对应创建 1 个 PV。每个 PD 会使用一个 PV。

- 给监控数据使用的盘，可以参考步骤挂载盘，创建目录，并将新建的目录以 bind mount 方式挂载到 /mnt/monitoring 目录下。

> 注意：
> 该步骤中创建的目录个数取决于规划的 TiDB 集群数量。1 个目录会对应创建 1 个 PV。每个 TiDB 集群的监控数据会使用 1 个 PV。

- 给 TiDB Binlog 和备份数据使用的盘，可以参考步骤挂载盘，创建目录，并将新建的目录以 bind mount 方式挂载到 /mnt/backup 目录下。

> 注意：
> 该步骤中创建的目录个数取决于规划的 TiDB 集群数量、每个集群内的 Pump 数量及备份方式。1 个目录会对应创建 1 个 PV。每个 Pump 会使用 1 个 PV，每个 drainer 会使用 1 个 PV，所有Ad-hoc 全量备份和所有定时全量备份会共用 1 个 PV。

上述的 /mnt/ssd、/mnt/sharedssd、/mnt/monitoring 和 /mnt/backup 是 local-volume-provisioner 使用的发现目录 (discovery directory)，local-volume-provisioner 会为发现目录下的每一个子目录创建对应的 PV。

第 2 步：部署 local-volume-provisioner

在线部署

1. 下载 local-volume-provisioner 部署文件。

```
wget https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ examples/local-pv/local-volume-provisioner.yaml
```

2. 如果你使用的发现路径与第 1 步：准备本地存储中的示例一致，可跳过这一步。如果你使用与上一步中不同路径的发现目录，需要修改 ConfigMap 和 DaemonSet 定义。

   - 修改 ConfigMap 定义中的 data.storageClassMap 字段：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-provisioner-config
  namespace: kube-system
data:
  # ...
  storageClassMap: |
    ssd-storage: # 给 TiKV 使用
      hostDir: /mnt/ssd
      mountDir: /mnt/ssd
    shared-ssd-storage: # 给 PD 使用
      hostDir: /mnt/sharedssd
      mountDir: /mnt/sharedssd
    monitoring-storage: # 给监控数据使用
      hostDir: /mnt/monitoring
      mountDir: /mnt/monitoring
    backup-storage: # 给 TiDB Binlog 和备份数据使用
      hostDir: /mnt/backup
      mountDir: /mnt/backup
```

   关于 local-volume-provisioner 更多的配置项，参考文档 Configuration 。

   - 修改 DaemonSet 定义中的 volumes 与 volumeMounts 字段，以确保发现目录能够挂载到 Pod 中的对应目录：

```
......
      volumeMounts:
        - mountPath: /mnt/ssd
          name: local-ssd
          mountPropagation: "HostToContainer"
        - mountPath: /mnt/sharedssd
          name: local-sharedssd
          mountPropagation: "HostToContainer"
        - mountPath: /mnt/backup
          name: local-backup
          mountPropagation: "HostToContainer"
        - mountPath: /mnt/monitoring
          name: local-monitoring
          mountPropagation: "HostToContainer"
  volumes:
    - name: local-ssd
```

```
      hostPath:
        path: /mnt/ssd
    - name: local-sharedssd
      hostPath:
        path: /mnt/sharedssd
    - name: local-backup
      hostPath:
        path: /mnt/backup
    - name: local-monitoring
      hostPath:
        path: /mnt/monitoring
......
```

3. 部署 local-volume-provisioner 程序。

```
kubectl apply -f local-volume-provisioner.yaml
```

4. 检查 Pod 和 PV 状态。

```
kubectl get po -n kube-system -l app=local-volume-provisioner && \
kubectl get pv | grep -e ssd-storage -e shared-ssd-storage -e
    ↪ monitoring-storage -e backup-storage
```

local-volume-provisioner 会为发现目录下的每一个挂载点创建一个 PV。

> 注意：
> 如果发现目录下无任何挂载点，则不会创建任何 PV，那么输出将为空。

更多信息，可参阅 Kubernetes 本地存储和 local-static-provisioner 文档。

离线部署

离线部署步骤与在线部署步骤相同，需要注意的是：

- 先在有外网的服务器下载 local-volume-provisioner 部署文件，上传到服务器上后再进行安装。
- local-volume-provisioner 程序是一个 DaemonSet，会在每个 Kubernetes 工作节点上启动一个 Pod，这个 Pod 使用的镜像是 quay.io/external_storage/local-volume
  ↪ -provisioner:v2.3.4，如果服务器没有外网，需要先将此 Docker 镜像在有外网的机器下载下来：

  shell docker pull quay.io/external_storage/local-volume-provisioner:
  ↪ v2.3.4 docker save -o local-volume-provisioner-v2.3.4.tar quay.io/
  ↪ external_storage/local-volume-provisioner:v2.3.4

将 local-volume-provisioner-v2.3.4.tar 文件拷贝到服务器上，执行 docker load 命令将其 load 到服务器上：

```
shell docker load -i local-volume-provisioner-v2.3.4.tar
```

最佳实践

- 本地 PV 的路径是本地存储卷的唯一标示符。为了保证唯一性并避免冲突，推荐使用设备的 UUID 来生成唯一的路径。
- 如果想要 IO 隔离，建议每个存储卷使用一块物理盘，在硬件层隔离。
- 如果想要容量隔离，建议每个存储卷一个分区使用一块物理盘，或者每个存储卷使用一块物理盘。

更多信息，可参阅 local-static-provisioner 的最佳实践文档。

4.1.5.2.4 数据安全

一般情况下 PVC 在使用完删除后，与其绑定的 PV 会被 provisioner 清理回收再放入资源池中被调度使用。为避免数据意外丢失，可在全局配置 StorageClass 的回收策略 (reclaim policy) 为 Retain 或者只将某个 PV 的回收策略修改为 Retain。Retain 模式下，PV 不会自动被回收。

- 全局配置

  StorageClass 的回收策略一旦创建就不能再修改，所以只能在创建时进行设置。如果创建时没有设置，可以再创建相同 provisioner 的 StorageClass，例如 GKE 上默认的 pd 类型的 StorageClass 默认保留策略是 Delete，可以再创建一个名为 pd-standard 的保留策略是 Retain 的存储类型，并在创建 TiDB 集群时将相应组件的 storageClassName 修改为 pd-standard。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: pd-standard
parameters:
  type: pd-standard
provisioner: kubernetes.io/gce-pd
reclaimPolicy: Retain
volumeBindingMode: Immediate
```

- 配置单个 PV

```
kubectl patch pv ${pv_name} -p '{"spec":{"persistentVolumeReclaimPolicy
    ↪ ":"Retain"}}'
```

> **注意：**
>
> TiDB Operator 默认会自动将 PD 和 TiKV 的 PV 保留策略修改为 `Retain` 以确保数据安全。

删除 PV 以及对应的数据

PV 保留策略是 `Retain` 时，如果确认某个 PV 的数据可以被删除，需要严格按照下面的操作顺序来删除 PV 以及对应的数据：

1. 删除 PV 对应的 PVC 对象：

```
kubectl delete pvc ${pvc_name} --namespace=${namespace}
```

2. 设置 PV 的保留策略为 `Delete`，PV 会被自动删除并回收：

```
kubectl patch pv ${pv_name} -p '{"spec":{"persistentVolumeReclaimPolicy
    ↪ ":"Delete"}}'
```

要了解更多关于 PV 的保留策略可参考修改 PV 保留策略。

4.1.5.3 在 Kubernetes 上部署 TiDB Operator

本文介绍如何在 Kubernetes 上部署 TiDB Operator。

4.1.5.3.1 准备环境

TiDB Operator 部署前，请确认以下软件需求：

- Kubernetes v1.12 或者更高版本
- DNS 插件
- PersistentVolume
- RBAC 启用（可选）
- Helm 3

4.1.5.3.2 部署 Kubernetes 集群

TiDB Operator 运行在 Kubernetes 集群，你可以使用 Getting started 页面列出的任何一种方法搭建一套 Kubernetes 集群。只要保证 Kubernetes 版本大于等于 v1.12。若想创建一个简单集群测试，可以参考快速上手教程。

对于部分公有云环境，可以参考如下文档部署 TiDB Operator 及 TiDB 集群：

- 部署到 AWS EKS
- 部署到 GCP GKE
- 部署到阿里云 ACK

TiDB Operator 使用持久化卷持久化存储 TiDB 集群数据（包括数据库，监控和备份数据），所以 Kubernetes 集群必须提供至少一种持久化卷。为提高性能，建议使用本地 SSD 盘作为持久化卷。可以根据这一步配置本地持久化卷。

Kubernetes 集群建议启用 RBAC。

4.1.5.3.3 安装 Helm

参考使用 Helm 安装 Helm 并配置 PingCAP 官方 chart 仓库。

4.1.5.3.4 配置本地持久化卷

参考本地 PV 配置在你的 Kubernetes 集群中配置本地持久化卷。

4.1.5.3.5 部署 TiDB Operator

创建 CRD

TiDB Operator 使用 Custom Resource Definition (CRD) 扩展 Kubernetes，所以要使用 TiDB Operator，必须先创建 `TidbCluster` 自定义资源类型。只需要在你的 Kubernetes 集群上创建一次即可：

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-operator/v1
    ↪ .1.15/manifests/crd.yaml
```

如果服务器没有外网，需要先用有外网的机器下载 `crd.yaml` 文件，然后再进行安装：

```
wget https://raw.githubusercontent.com/pingcap/tidb-operator/v1.1.15/
    ↪ manifests/crd.yaml
kubectl apply -f ./crd.yaml
```

如果显示如下信息表示 CRD 安装成功：

```
kubectl get crd
```

```
NAME                                  CREATED AT
backups.pingcap.com                   2020-06-11T07:59:40Z
backupschedules.pingcap.com           2020-06-11T07:59:41Z
restores.pingcap.com                  2020-06-11T07:59:40Z
tidbclusterautoscalers.pingcap.com 2020-06-11T07:59:42Z
tidbclusters.pingcap.com              2020-06-11T07:59:38Z
tidbinitializers.pingcap.com          2020-06-11T07:59:42Z
tidbmonitors.pingcap.com              2020-06-11T07:59:41Z
```

自定义部署 TiDB Operator

若需要快速部署 TiDB Operator，可参考快速上手中部署 TiDB Operator 文档。本节介绍自定义部署 TiDB Operator 的配置方式。

创建 CRDs 之后，在 Kubernetes 集群上部署 TiDB Operator 有两种方式：在线和离线部署。

在线部署 TiDB Operator

1. 获取你要部署的 tidb-operator chart 中的 values.yaml 文件：

```
mkdir -p ${HOME}/tidb-operator && \
helm inspect values pingcap/tidb-operator --version=${chart_version} >
    ↪ ${HOME}/tidb-operator/values-tidb-operator.yaml
```

> 注意：
> ${chart_version} 在后续文档中代表 chart 版本，例如 v1.1.15，可以通过 helm search repo -l tidb-operator 查看当前支持的版本。

2. 配置 TiDB Operator

   TiDB Operator 里面会用到 k8s.gcr.io/kube-scheduler 镜像，如果无法下载该镜像，可以修改 ${HOME}/tidb-operator/values-tidb-operator.yaml 文件中的 scheduler.kubeSchedulerImageName 为 registry.cn-hangzhou.aliyuncs.com/google_containers/kube-scheduler。

   其他项目例如：limits、requests 和 replicas，请根据需要进行修改。

3. 部署 TiDB Operator

```
helm install tidb-operator pingcap/tidb-operator --namespace=tidb-admin
    ↪  --version=${chart_version} -f ${HOME}/tidb-operator/values-tidb-
    ↪ operator.yaml && \
kubectl get po -n tidb-admin -l app.kubernetes.io/name=tidb-operator
```

> 注意：
> 如果对应 tidb-admin namespace 不存在，则可先使用 kubectl create namespace tidb-admin 创建该 namespace。

4. 升级 TiDB Operator

   如果需要升级 TiDB Operator，请先修改 ${HOME}/tidb-operator/values-tidb-operator.yaml 文件，然后执行下面的命令进行升级：

```
helm upgrade tidb-operator pingcap/tidb-operator --namespace=tidb-admin
    ↪  -f ${HOME}/tidb-operator/values-tidb-operator.yaml
```

离线安装 TiDB Operator

如果服务器没有外网，需要按照下面的步骤来离线安装 TiDB Operator：

1. 下载 tidb-operator chart

   如果服务器上没有外网，就无法通过配置 Helm repo 来安装 TiDB Operator 组件以及其他应用。这时，需要在有外网的机器上下载集群安装需用到的 chart 文件，再拷贝到服务器上。

   通过以下命令，下载 tidb-operator chart 文件：

   ```
   wget http://charts.pingcap.org/tidb-operator-v1.1.15.tgz
   ```

   将 tidb-operator-v1.1.15.tgz 文件拷贝到服务器上并解压到当前目录：

   ```
   tar zxvf tidb-operator.v1.1.15.tgz
   ```

2. 下载 TiDB Operator 运行所需的 Docker 镜像

   如果服务器没有外网，需要在有外网的机器上将 TiDB Operator 用到的所有 Docker 镜像下载下来并上传到服务器上，然后使用 docker load 将 Docker 镜像安装到服务器上。

   TiDB Operator 用到的 Docker 镜像有：

   ```
   pingcap/tidb-operator:v1.1.15
   pingcap/tidb-backup-manager:v1.1.15
   bitnami/kubectl:latest
   pingcap/advanced-statefulset:v0.3.3
   k8s.gcr.io/kube-scheduler:v1.16.9
   ```

   其中 k8s.gcr.io/kube-scheduler:v1.16.9 请跟你的 Kubernetes 集群的版本保持一致即可，不用单独下载。

   接下来通过下面的命令将所有这些镜像下载下来：

   ```
   docker pull pingcap/tidb-operator:v1.1.15
   docker pull pingcap/tidb-backup-manager:v1.1.15
   docker pull bitnami/kubectl:latest
   docker pull pingcap/advanced-statefulset:v0.3.3

   docker save -o tidb-operator-v1.1.15.tar pingcap/tidb-operator:v1.1.15
   docker save -o tidb-backup-manager-v1.1.15.tar pingcap/tidb-backup-
       ↪ manager:v1.1.15
   docker save -o bitnami-kubectl.tar bitnami/kubectl:latest
   ```

```
docker save -o advanced-statefulset-v0.3.3.tar pingcap/advanced-
    ↪ statefulset:v0.3.3
```

接下来将这些 Docker 镜像上传到服务器上，并执行 `docker load` 将这些 Docker 镜像安装到服务器上：

```
docker load -i tidb-operator-v1.1.15.tar
docker load -i tidb-backup-manager-v1.1.15.tar
docker load -i bitnami-kubectl.tar
docker load -i advanced-statefulset-v0.3.3.tar
```

3. 配置 TiDB Operator

   TiDB Operator 内嵌了一个 `kube-scheduler` 用来实现自定义调度器，请修改 `./tidb-operator/values.yaml` 文件来配置这个内置 `kube-scheduler` 组件的 Docker 镜像名字和版本，例如你的 Kubernetes 集群中的 `kube-scheduler` 使用的镜像为 `k8s.gcr.io/kube-scheduler:v1.16.9`，请这样设置 `./tidb-operator/values.yaml`：

```
...
scheduler:
  serviceAccount: tidb-scheduler
  logLevel: 2
  replicas: 1
  schedulerName: tidb-scheduler
  resources:
    limits:
      cpu: 250m
      memory: 150Mi
    requests:
      cpu: 80m
      memory: 50Mi
  kubeSchedulerImageName: k8s.gcr.io/kube-scheduler
  kubeSchedulerImageTag: v1.16.9
...
```

   其他项目例如：`limits`、`requests` 和 `replicas`，请根据需要进行修改。

4. 安装 TiDB Operator

   使用下面的命令安装 TiDB Operator：

```
helm install tidb-operator ./tidb-operator --namespace=tidb-admin
```

> 注意：
> 如果对应 `tidb-admin` namespace 不存在，则可先使用 `kubectl create namespace tidb-admin` 创建该 namespace。

5. 升级 TiDB Operator

   如果需要升级 TiDB Operator，请先修改 `./tidb-operator/values.yaml` 文件，然后执行下面的命令进行升级：

```
helm upgrade tidb-operator ./tidb-operator --namespace=tidb-admin
```

4.1.5.3.6 自定义配置 TiDB Operator

可以通过修改 `${HOME}/tidb-operator/values-tidb-operator.yaml` 来配置 TiDB Operator。本节后续使用 `values.yaml` 来代表 `${HOME}/tidb-operator/values-tidb-operator.yaml`。

TiDB Operator 包含两个组件：

- tidb-controller-manager
- tidb-scheduler

这两个组件都是无状态的，由 `Deployment` 部署。在 `values.yaml` 文件中，你可以配置其中的 `limit`、`request` 与 `replicas` 参数。

修改了 `values.yaml` 文件后，请运行以下命令使更改生效：

```
helm upgrade tidb-operator pingcap/tidb-operator --version=${chart_version}
    ↪ --namespace=tidb-admin -f ${HOME}/tidb-operator/values-tidb-operator.
    ↪ yaml
```

4.1.5.4 在 Kubernetes 中配置 TiDB 集群

本文档介绍了如何配置生产可用的 TiDB 集群。涵盖以下内容：

- 资源配置
- 部署配置
- 高可用配置

4.1.5.4.1 资源配置

部署前需要根据实际情况和需求，为 TiDB 集群各个组件配置资源，其中 PD、TiKV、TiDB 是 TiDB 集群的核心服务组件，在生产环境下它们的资源配置还需要按组件要求指定，具体参考：资源配置推荐。

为了保证 TiDB 集群的组件在 Kubernetes 中合理的调度和稳定的运行，建议为其设置 Guaranteed 级别的 QoS，通过在配置资源时让 limits 等于 requests 来实现，具体参考：配置 QoS。

如果使用 NUMA 架构的 CPU，为了获得更好的性能，需要在节点上开启 `Static` 的 CPU 管理策略。为了 TiDB 集群组件能独占相应的 CPU 资源，除了为其设置上述 Guaranteed 级别的 QoS 外，还需要保证 CPU 的配额必须是大于或等于 1 的整数。具体参考：CPU 管理策略。

4.1.5.4.2 部署配置

通过配置 TidbCluster CR 来配置 TiDB 集群。参考 TidbCluster 示例和 API 文档（示例和 API 文档请切换到当前使用的 TiDB Operator 版本）完成 TidbCluster CR (Custom Resource)。

> **注意：**
>
> 建议在 ${cluster_name} 目录下组织 TiDB 集群的配置，并将其另存为 ${cluster_name}/tidb-cluster.yaml。默认条件下，修改配置不会自动应用到 TiDB 集群中，只有在 Pod 重启时，才会重新加载新的配置文件。

集群名称

通过更改 TiDBCuster CR 中的 metadata.name 来配置集群名称。

版本

正常情况下，集群内的各组件应该使用相同版本，所以一般建议配置 spec.<pd/tidb/tikv/pump/tiflash/ticdc>.baseImage + spec.version 即可。如果需要为不同的组件配置不同的版本，则可以配置 spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version。

相关参数的格式如下：

- spec.version，格式为 imageTag，例如 v5.0.6
- spec.<pd/tidb/tikv/pump/tiflash/ticdc>.baseImage，格式为 imageName，例如 pingcap/tidb
- spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version，格式为 imageTag，例如 v5.0.6

推荐配置

configUpdateStrategy

建议设置 spec.configUpdateStrategy: RollingUpdate，开启配置自动更新特性，在每次配置更新时，自动对组件执行滚动更新，将修改后的配置应用到集群中。

enableDynamicConfiguration

建议通过设置 spec.enableDynamicConfiguration: true 配置 TiKV 的 --advertise-status-addr 启动参数。

版本支持：TiDB v4.0.1 及更高版本，TiDB Operator v1.1.1 及更高版本。

pvReclaimPolicy

建议设置 spec.pvReclaimPolicy: Retain，确保 PVC 被删除后 PV 仍然保留，保证数据安全。

mountClusterClientSecret

PD 和 TiKV 支持配置 mountClusterClientSecret。如果开启了集群组件间 TLS 支持，建议配置 spec.pd.mountClusterClientSecret: true 和 spec.tikv.mountClusterClientSecret: true，这样 TiDB Operator 会自动将 ${cluster_name}-cluster-client-secret 证书挂载到 PD 和 TiKV 容器，方便使用 pd-ctl 和 tikv-ctl。

存储

存储类型

如果需要设置存储类型，可以修改 ${cluster_name}/tidb-cluster.yaml 中各组件的 storageClassName 字段。关于 Kubernetes 集群支持哪些存储类型，请联系系统管理员确定。

另外，TiDB 集群不同组件对磁盘的要求不一样，所以部署集群前，要根据当前 Kubernetes 集群支持的存储类型以及使用场景，参考存储配置文档为 TiDB 集群各组件选择合适的存储类型。

> 注意：
>
> 如果创建 TiDB 集群时设置了 Kubernetes 集群中不存在的存储类型，则会导致 TiDB 集群创建处于 Pending 状态，需要将 TiDB 集群彻底销毁掉，再进行重试。

多盘挂载

TiDB Operator 支持为 PD、TiDB、TiKV、TiCDC 挂载多块 PV，可以用于不同用途的数据写入。

每个组件都可以配置 storageVolumes 字段，用于描述用户自定义的多个 PV。

> 注意：
>
> 你需要在集群创建之前配置 storageVolumes。集群创建完成后，不支持添加或者删除 storageVolumes。对于已经配置的 storageVolumes，除增大 storageVolume.storageSize 外，其他项不支持修改。如果要增大 storageVolume.storageSize，需要对应的 StorageClass 支持动态扩容。

相关字段的含义如下：

- storageVolume.name：PV 的名称。

- `storageVolume.storageClassName`：PV 使用哪一个 StorageClass。如果不配置，会使用 spec.pd/tidb/tikv.storageClassName。
- `storageVolume.storageSize`：申请 PV 存储容量的大小。
- `storageVolume.mountPath`：将 PV 挂载到容器的哪个目录。

例子：

```
pd:
  baseImage: pingcap/pd
  replicas: 1
  # if storageClassName is not set, the default Storage Class of the
     ↪ Kubernetes cluster will be used
  # storageClassName: local-storage
  requests:
    storage: "1Gi"
  config:
    log:
      file:
        filename: /var/log/pdlog/pd.log
      level: "warn"
  storageVolumes:
  - name: log
    storageSize: "2Gi"
    mountPath: "/var/log/pdlog"
tidb:
  baseImage: pingcap/tidb
  replicas: 1
  service:
    type: ClusterIP
  config:
    log:
      file:
        filename: /var/log/tidblog/tidb.log
      level: "warn"
  storageVolumes:
  - name: log
    storageSize: "2Gi"
    mountPath: "/var/log/tidblog"
tikv:
  baseImage: pingcap/tikv
  replicas: 1
  # if storageClassName is not set, the default Storage Class of the
     ↪ Kubernetes cluster will be used
  # storageClassName: local-storage
  requests:
```

```
    storage: "1Gi"
  config:
    storage:
      # In basic examples, you can set this to avoid using too much storage
          ↪ .
      reserve-space: "0MB"
    rocksdb:
      wal-dir: "/data_sbi/tikv/wal"
    titan:
      dirname: "/data_sbj/titan/data"
  storageVolumes:
  - name: wal
    storageSize: "2Gi"
    mountPath: "/data_sbi/tikv/wal"
  - name: titan
    storageSize: "2Gi"
    mountPath: "/data_sbj/titan/data"
```

注意：

TiDB Operator 默认会使用一些挂载路径，比如会为 TiDB Pod 挂载 EmptyDir 到 /var/log/tidb 目录。在配置 storageVolumes 的时候要避免配置重复的 mountPath。

HostNetwork

PD、TiKV、TiDB、TiFlash、TiCDC 及 Pump 支持配置 Pod 使用宿主机上的网络命名空间 HostNetwork。可通过配置 spec.hostNetwork: true 为所有受支持的组件开启，或通过为特定组件配置 hostNetwork: true 为单个或多个组件开启。

Discovery

TiDB Operator 会为每一个 TiDB 集群启动一个 Discovery 服务。Discovery 服务会为每个 PD Pod 返回相应的启动参数，来辅助 PD 集群启动。你可以通过 spec.discovery 配置 Discovery 服务的资源，详见容器资源管理。

spec.discovery 配置示例：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v4.0.10
```

```
  pvReclaimPolicy: Retain
  discovery:
    limits:
      cpu: "0.2"
    requests:
      cpu: "0.2"
  pd:
    baseImage: pingcap/pd
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
...
```

集群拓扑

PD/TiKV/TiDB

默认示例的集群拓扑是：3 个 PD Pod，3 个 TiKV Pod，2 个 TiDB Pod。在该部署拓扑下根据数据高可用原则，TiDB Operator 扩展调度器要求 Kubernetes 集群中至少有 3 个节点。可以修改 replicas 配置来更改每个组件的 Pod 数量。

> **注意：**
>
> 如果 Kubernetes 集群节点个数少于 3 个，将会导致有一个 PD Pod 处于 Pending 状态，而 TiKV 和 TiDB Pod 也都不会被创建。Kubernetes 集群节点个数少于 3 个时，为了使 TiDB 集群能启动起来，可以将默认部署的 PD Pod 个数减小到 1 个。

部署 TiFlash

如果要在集群中开启 TiFlash，需要在 ${cluster_name}/tidb-cluster.yaml 文件中配置 spec.pd.config.replication.enable-placement-rules: true，并配置 spec.tiflash:

```
  pd:
    config: |
      ...
      [replication]
      enable-placement-rules = true
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 3
    replicas: 1
```

```
  storageClaims:
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
```

TiFlash 支持挂载多个 PV，如果要为 TiFlash 配置多个 PV，可以在 tiflash.storageClaims 下面配置多项，每一项可以分别配置 storage reqeust 和 storageClassName，例如：

```
tiflash:
  baseImage: pingcap/tiflash
  maxFailoverCount: 3
  replicas: 1
  storageClaims:
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
```

所有 PV 按照配置先后顺序分别挂载到容器内的 /data0、/data1 等目录。TiFlash 有 4 个日志文件，其中 Proxy 日志打印到容器标准输出，另外 3 个日志存储在硬盘中，默认存储在 /data0 目录下，分别为 /data0/logs/flash_cluster_manager.log、/data0/logs/error.log、/data0/logs/server.log，如果要修改日志存储路径，可以参考配置 TiFlash 配置参数进行修改。

> **警告：**
>
> 由于 TiDB Operator 会按照 storageClaims 列表中的配置按顺序自动挂载 PV，如果需要为 TiFlash 增加磁盘，请确保只在列表原有配置最后添加，并且不能修改列表中原有配置的顺序。

部署 TiCDC

如果要在集群中开启 TiCDC，需要在 ${cluster_name}/tidb-cluster.yaml 文件中配置 spec.ticdc：

```
ticdc:
  baseImage: pingcap/ticdc
  replicas: 3
```

```
    config:
      logLevel: info
```

配置 TiDB 组件

本节介绍如何配置 TiDB/TiKV/PD/TiFlash/TiCDC 的配置选项。

配置 TiDB 配置参数

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

对于 TiDB Operator v1.1.6 及之后版本，请使用 TOML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
....
  tidb:
    image: pingcap/tidb:v5.0.6
    imagePullPolicy: IfNotPresent
    replicas: 1
    service:
      type: ClusterIP
    config: |
      split-table = true
      oom-action = "log"
    requests:
      cpu: 1
```

对于 TiDB Operator v1.1.6 之前版本，请使用 YAML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
....
  tidb:
    image: pingcap/tidb:v5.0.6
    imagePullPolicy: IfNotPresent
    replicas: 1
    service:
      type: ClusterIP
    config:
      split-table: true
      oom-action: "log"
```

```
    requests:
      cpu: 1
```

获取所有可以配置的 TiDB 配置参数，请参考 TiDB 配置文档。

> **注意：**
>
> 为了兼容 `helm` 部署，如果你是通过 CR 文件部署 TiDB 集群，即使你不设置 Config 配置，也需要保证 `Config: {}` 的设置，从而避免 TiDB 组件无法正常启动。

配置 TiKV 配置参数

你可以通过 TidbCluster CR 的 `spec.tikv.config` 来配置 TiKV 配置参数。

对于 TiDB Operator v1.1.6 及之后版本，请使用 TOML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
....
  tikv:
    image: pingcap/tikv:v5.0.1
    config: |
      [storage]
        [storage.block-cache]
          capacity = "16GB"
    replicas: 1
    requests:
      cpu: 2
```

对于 TiDB Operator v1.1.6 之前版本，请使用 YAML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
....
  tikv:
    image: pingcap/tikv:v5.0.1
    config:
```

```
    storage:
      block-cache:
        capacity: "16GB"
  replicas: 1
  requests:
    cpu: 2
```

获取所有可以配置的 TiKV 配置参数，请参考 TiKV 配置文档

注意：

为了兼容 `helm` 部署，如果你是通过 CR 文件部署 TiDB 集群，即使你不设置 Config 配置，也需要保证 `Config: {}` 的设置，从而避免 TiKV 组件无法正常启动。

配置 PD 配置参数

你可以通过 TidbCluster CR 的 `spec.pd.config` 来配置 PD 配置参数。

对于 TiDB Operator v1.1.6 及之后版本，请使用 TOML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
.....
  pd:
    image: pingcap/pd:v5.0.1
    config: |
      lease = 3
      enable-prevote = true
```

对于 TiDB Operator v1.1.6 之前版本，请使用 YAML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
.....
  pd:
    image: pingcap/pd:v5.0.1
    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：

- 为了兼容 `helm` 部署，如果你是通过 CR 文件部署 TiDB 集群，即使你不设置 Config 配置，也需要保证 `Config: {}` 的设置，从而避免 PD 组件无法正常启动。
- 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。

配置 TiFlash 配置参数

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

对于 TiDB Operator v1.1.6 及之后版本，请使用 TOML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  ...
  tiflash:
    config:
      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"
```

对于 TiDB Operator v1.1.6 之前版本，请使用 YAML 格式配置：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  ...
  tiflash:
    config:
      config:
        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

配置 TiCDC 启动参数

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

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  ...
  ticdc:
    config:
      timezone: UTC
      gcTTL: 86400
      logLevel: 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。

配置 PD、TiDB、TiKV、TiFlash 故障自动转移阈值

故障自动转移功能在 TiDB Operator 中默认开启。当 PD、TiDB、TiKV、TiFlash 这些组件的 Pod 或者其所在节点发生故障时，TiDB Operator 会触发故障自动转移，通过扩容相应组件补齐 Pod 副本数。

为避免故障自动转移功能创建太多 Pod，可以为每个组件配置故障自动转移时能扩容的 Pod 数量阈值，默认为 3。如果配置为 0，代表关闭这个组件的故障自动转移功能。配置示例如下：

```
  pd:
    maxFailoverCount: 3
  tidb:
    maxFailoverCount: 3
  tikv:
    maxFailoverCount: 3
  tiflash:
    maxFailoverCount: 3
```

配置 TiDB 平滑升级

滚动更新 TiDB 集群的过程中，在停止 TiDB Pod 之前，Kubernetes 会向 TiDB server 进程发送一个 TERM 信号。在收到 TERM 信号后，TiDB server 会尝试等待所有的连接关闭，不过 15 秒后会强制关闭所有连接并退出进程。

从 v1.1.2 版本开始，TiDB Operator 已经支持平滑升级 TiDB 集群。通过配置下面两个属性来实现平滑升级 TiDB 集群：

- spec.tidb.terminationGracePeriodSeconds：滚动更新的时候，删除旧的 TiDB Pod 最多容忍的时间，即过了这个时间，TiDB Pod 会被强制删除；
- spec.tidb.lifecycle：设置 TiDB Pod 的 preStop Hook，在 TiDB server 停止之前执行的操作。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v5.0.6
  pvReclaimPolicy: Retain
  discovery: {}
  pd:
    baseImage: pingcap/pd
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
```

```
  replicas: 1
  service:
    type: ClusterIP
  config: {}
  terminationGracePeriodSeconds: 60
  lifecycle:
    preStop:
      exec:
        command:
        - /bin/sh
        - -c
        - "sleep 10 && kill -QUIT 1"
```

上述 YAML 文件中：

- 设置了删除 TiDB Pod 的最多容忍时间为 60 秒，如果 60 秒之内客户端仍然没有关闭连接的话，那么这些连接将会强制关闭。这个时间可根据需要进行调整；
- 设置 `preStop` Hook 为 `sleep 10 && kill -QUIT 1`，这里 Pid 1 为 TiDB Pod 内 TiDB server 进程的 Pid。TiDB server 进程收到这个信号之后，会等待所有连接被客户端关闭之后才会退出。

Kubernetes 在删除 TiDB Pod 的同时，也会把该 TiDB 节点从 Service 的 Endpoints 中移除。这样就可以保证新的连接不会连接到该 TiDB 节点，但是由于此过程是异步的，所以可以在发送 Kill 信号之前 sleep 几秒钟，确保该 TiDB 节点从 Endpoints 中去掉。

配置 TiKV 平滑升级

TiKV 升级过程中，在重启 TiKV Pod 之前，TiDB Operator 会先驱逐 TiKV Pod 上的所有 Region leader。只有当驱逐完成（即 TiKV Pod 上的 Region leader 个数为 0）或者驱逐超时（默认 10 分钟）后，TiKV Pod 才会重启。

如果驱逐 Region leader 超时，重启 TiKV Pod 会导致部分请求失败或者延时增加。要避免此问题，你可以将超时时间 `spec.tikv.evictLeaderTimeout`（默认 10 分钟）配置为一个更大的值，例如：

```
spec:
  tikv:
    evictLeaderTimeout: 10000m
```

> **警告：**
>
> 如果使用 TiKV 版本小于 4.0.14，或者小于 5.0.3，由于 TiKV 的 bug，需要将 `spec.tikv.evictLeaderTimeout` 的值设置的尽可能大（推荐大于 `1500m`），以保证 TiKV Pod 上所有的 Region Leader 能在设置的时间内驱逐完毕。

配置 TiDB 慢查询日志持久卷

默认配置下，TiDB Operator 会新建名称为 slowlog 的 EmptyDir 卷来存储慢查询日志，slowlog 卷默认挂载到 /var/log/tidb，慢查询日志通过 sidecar 容器打印到标准输出。

> **警告：**
>
> 默认配置下，使用 EmptyDir 卷存储的慢查询日志会在 Pod 被删除（例如，滚动升级）后丢失。请确保 Kubernetes 集群内已经部署日志收集方案用于收集所有容器的日志。如果没有部署日志收集方案，请务必通过下面配置使用持久卷来存储慢查询日志。

如果想使用单独的持久卷来存储慢查询日志，可以通过配置 spec.tidb.slowLogVolumeName 单独指定存储慢查询日志的持久卷名称，并在 spec.tidb.storageVolumes 或 spec.tidb.additionalVolumes 配置持久卷信息。下面分别演示使用 spec.tidb.storageVolumes 和 spec.tidb.additionalVolumes 配置持久卷。

spec.tidb.storageVolumes 配置

按照如下示例配置 TidbCluster CR，TiDB Operator 将使用持久卷 ${volumeName} 存储慢查询日志，日志文件路径为：${mountPath}/${volumeName}。spec.tidb.storageVolumes 字段的具体配置方式可参考多盘挂载。

```
tidb:
  ...
  separateSlowLog: true # 可省略
  slowLogVolumeName: ${volumeName}
  storageVolumes:
    # name 必须和 slowLogVolumeName 字段的值保持一致
    - name: ${volumeName}
      storageClassName: ${storageClass}
      storageSize: "1Gi"
      mountPath: ${mountPath}
```

spec.tidb.additionalVolumes 配置（从 v1.1.8 版本开始支持）

下面以 NFS 为例配置 spec.tidb.additionalVolumes。TiDB Operator 将使用持久卷 ${volumeName} 存储慢查询日志，日志文件路径为：${mountPath}/${volumeName}。具体支持的持久卷类型可参考 Persistent Volumes。

```
tidb:
  ...
  separateSlowLog: true # 可省略
  slowLogVolumeName: ${volumeName}
```

```
  additionalVolumes:
  # name 必须和 slowLogVolumeName 字段的值保持一致
  - name: ${volumeName}
    nfs:
      server: 192.168.0.2
      path: /nfs
  additionalVolumeMounts:
  # name 必须和 slowLogVolumeName 字段的值保持一致
  - name: ${volumeName}
    mountPath: ${mountPath}
```

配置 TiDB 服务

需要配置 spec.tidb.service，TiDB Operator 才会为 TiDB 创建 Service。Service 可以根据场景配置不同的类型，比如 ClusterIP、NodePort、LoadBalancer 等。

通用配置

不用类型的 Service 有着部分通用的配置，包括：

- spec.tidb.service.annotations：添加到 Service 资源的 Annotation。
- spec.tidb.service.labels：添加到 Service 资源的 Labels。

ClusterIP

ClusterIP 是通过集群的内部 IP 暴露服务，选择该类型的服务时，只能在集群内部访问，使用 ClusterIP 或者 Service 域名（${cluster_name}-tidb.${namespace}）访问。

```
spec:
  ...
  tidb:
    service:
      type: ClusterIP
```

NodePort

在没有 LoadBalancer 时，可选择通过 NodePort 暴露。NodePort 是通过节点的 IP 和静态端口暴露服务。通过请求 NodeIP + NodePort，可以从集群的外部访问一个 NodePort 服务。

```
spec:
  ...
  tidb:
    service:
      type: NodePort
      # externalTrafficPolicy: Local
```

NodePort 有两种模式：

- `externalTrafficPolicy=Cluster`：集群所有的机器都会给 TiDB 分配 NodePort 端口，此为默认值

  使用 `Cluster` 模式时，可以通过任意一台机器的 IP 加 NodePort 访问 TiDB 服务，如果该机器上没有 TiDB Pod，则相应请求会转发到有 TiDB Pod 的机器上。

> **注意：**
> 该模式下 TiDB 服务获取到的请求源 IP 是主机 IP，并不是真正的客户端源 IP，所以基于客户端源 IP 的访问权限控制在该模式下不可用。

- `externalTrafficPolicy=Local`：只有运行 TiDB 的机器会分配 NodePort 端口，用于访问本地的 TiDB 实例

LoadBalancer

若运行在有 LoadBalancer 的环境，比如 GCP/AWS 平台，建议使用云平台的 LoadBalancer 特性。

```
spec:
  ...
  tidb:
    service:
      annotations:
        cloud.google.com/load-balancer-type: "Internal"
      externalTrafficPolicy: Local
      type: LoadBalancer
```

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

4.1.5.4.3 高可用配置

> **注意：**
> TiDB Operator 提供了自定义的调度器，该调度器通过指定的调度算法能在 host 层面保证 TiDB 服务的高可用。目前，TiDB 集群使用该调度器作为默认调度器，可通过 `spec.schedulerName` 配置项进行设置。本节重点介绍如何配置 TiDB 集群以容忍其他级别的故障，例如机架、可用区或 region。本部分可根据使用需求配置，不是必选。

TiDB 是分布式数据库，它的高可用需要做到在任一个物理拓扑节点发生故障时，不仅服务不受影响，还要保证数据也是完整和可用。下面分别具体说明这两种高可用的配置。

TiDB 服务高可用

通过 nodeSelector 调度实例

通过各组件配置的 nodeSelector 字段，可以约束组件的实例只能调度到特定的节点上。关于 nodeSelector 的更多说明，请参阅 nodeSelector。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
#### ...
spec:
  pd:
    nodeSelector:
      node-role.kubernetes.io/pd: true
    # ...
  tikv:
    nodeSelector:
      node-role.kubernetes.io/tikv: true
    # ...
  tidb:
    nodeSelector:
      node-role.kubernetes.io/tidb: true
    # ...
```

通过 tolerations 调度实例

通过各组件配置的 tolerations 字段，可以允许组件的实例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。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
#### ...
spec:
  pd:
    tolerations:
      - effect: NoSchedule
        key: dedicated
        operator: Equal
        value: pd
    # ...
  tikv:
    tolerations:
      - effect: NoSchedule
        key: dedicated
        operator: Equal
        value: tikv
    # ...
  tidb:
```

```
  tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Equal
      value: tidb
  # ...
```

通过 affinity 调度实例

配置 PodAntiAffinity 能尽量避免同一组件的不同实例部署到同一个物理拓扑节点上，从而达到高可用的目的。关于 Affinity 的使用说明，请参阅 Affinity & AntiAffinity。

下面是一个典型的高可用设置例子：

```
affinity:
 podAntiAffinity:
   preferredDuringSchedulingIgnoredDuringExecution:
   # this term works when the nodes have the label named region
   - weight: 10
     podAffinityTerm:
       labelSelector:
         matchLabels:
           app.kubernetes.io/instance: ${cluster_name}
           app.kubernetes.io/component: "pd"
       topologyKey: "region"
       namespaces:
       - ${namespace}
   # this term works when the nodes have the label named zone
   - weight: 20
     podAffinityTerm:
       labelSelector:
         matchLabels:
           app.kubernetes.io/instance: ${cluster_name}
           app.kubernetes.io/component: "pd"
       topologyKey: "zone"
       namespaces:
       - ${namespace}
   # this term works when the nodes have the label named rack
   - weight: 40
     podAffinityTerm:
       labelSelector:
         matchLabels:
           app.kubernetes.io/instance: ${cluster_name}
           app.kubernetes.io/component: "pd"
       topologyKey: "rack"
       namespaces:
       - ${namespace}
```

```
  # this term works when the nodes have the label named kubernetes.io/
    ↪ hostname
  - weight: 80
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app.kubernetes.io/instance: ${cluster_name}
          app.kubernetes.io/component: "pd"
      topologyKey: "kubernetes.io/hostname"
      namespaces:
      - ${namespace}
```

数据的高可用

在开始数据高可用配置前，首先请阅读集群拓扑信息配置。该文档描述了 TiDB 集群数据高可用的实现原理。

在 Kubernetes 上支持数据高可用的功能，需要如下操作：

- 为 PD 设置拓扑位置 Label 集合

  用 Kubernetes 集群 Node 节点上描述拓扑位置的 Label 集合替换 `pd.config` 配置项中里的 `location-labels` 信息。

  > 注意：
  >
  > - PD 版本 < v3.0.9 不支持名字中带 `/` 的 Label。
  > - 如果在 `location-labels` 中配置 `host`，TiDB Operator 会从 Node Label 中的 `kubernetes.io/hostname` 获取值。

- 为 TiKV 节点设置所在的 Node 节点的拓扑信息

  TiDB Operator 会自动为 TiKV 获取其所在 Node 节点的拓扑信息，并调用 PD 接口将这些信息设置为 TiKV 的 store labels 信息，这样 TiDB 集群就能基于这些信息来调度数据副本。

  如果当前 Kubernetes 集群的 Node 节点没有表示拓扑位置的 Label，或者已有的拓扑 Label 名字中带有 `/`，可以通过下面的命令手动给 Node 增加标签：

  ```
  kubectl label node ${node_name} region=${region_name} zone=${zone_name}
    ↪  rack=${rack_name} kubernetes.io/hostname=${host_name}
  ```

  其中 `region`、`zone`、`rack`、`kubernetes.io/hostname` 只是举例，要添加的 Label 名字和数量可以任意定义，只要符合规范且和 `pd.config` 里的 `location-labels` 设置的 Labels 保持一致即可。

4.1.5.5 在标准 Kubernetes 上部署 TiDB 集群

本文主要描述了如何在标准的 Kubernetes 集群上通过 TiDB Operator 部署 TiDB 集群。

4.1.5.5.1 前置条件

- TiDB Operator 部署完成。

4.1.5.5.2 部署 TiDB 集群

在部署 TiDB 集群之前，需要先配置 TiDB 集群。请参阅在 Kubernetes 中配置 TiDB 集群。

配置 TiDB 集群后，请按照以下步骤部署 TiDB 集群：

1. 创建 Namespace：

```
kubectl create namespace ${namespace}
```

> 注意：
> namespace 是命名空间，可以起一个方便记忆的名字，比如和 cluster_name 相同的名称。

2. 部署 TiDB 集群：

```
kubectl apply -f ${cluster_name} -n ${namespace}
```

> 注意：
> 建议在 cluster_name 目录下组织 TiDB 集群的配置，并将其另存为 ${cluster_name}/tidb-cluster.yaml。默认条件下，修改配置不会自动应用到 TiDB 集群中，只有在 Pod 重启时，才会重新加载新的配置文件。

如果服务器没有外网，需要在有外网的机器上将 TiDB 集群用到的 Docker 镜像下载下来并上传到服务器上，然后使用 docker load 将 Docker 镜像安装到服务器上。

部署一套 TiDB 集群会用到下面这些 Docker 镜像（假设 TiDB 集群的版本是 v5.0.6）：

```
pingcap/pd:v5.0.6
pingcap/tikv:v5.0.6
pingcap/tidb:v5.0.6
```

```
pingcap/tidb-binlog:v5.0.6
pingcap/ticdc:v5.0.6
pingcap/tiflash:v5.0.6
pingcap/tidb-monitor-reloader:v1.0.1
pingcap/tidb-monitor-initializer:v5.0.6
grafana/grafana:6.0.1
prom/prometheus:v2.18.1
busybox:1.26.2
```

接下来通过下面的命令将所有这些镜像下载下来：

```
docker pull pingcap/pd:v5.0.6
docker pull pingcap/tikv:v5.0.6
docker pull pingcap/tidb:v5.0.6
docker pull pingcap/tidb-binlog:v5.0.6
docker pull pingcap/ticdc:v5.0.6
docker pull pingcap/tiflash:v5.0.6
docker pull pingcap/tidb-monitor-reloader:v1.0.1
docker pull pingcap/tidb-monitor-initializer:v5.0.6
docker pull grafana/grafana:6.0.1
docker pull prom/prometheus:v2.18.1
docker pull busybox:1.26.2

docker save -o pd-v5.0.6.tar pingcap/pd:v5.0.6
docker save -o tikv-v5.0.6.tar pingcap/tikv:v5.0.6
docker save -o tidb-v5.0.6.tar pingcap/tidb:v5.0.6
docker save -o tidb-binlog-v5.0.6.tar pingcap/tidb-binlog:v5.0.6
docker save -o ticdc-v5.0.6.tar pingcap/ticdc:v5.0.6
docker save -o tiflash-v5.0.6.tar pingcap/tiflash:v5.0.6
docker save -o tidb-monitor-reloader-v1.0.1.tar pingcap/tidb-monitor-
   ↪ reloader:v1.0.1
docker save -o tidb-monitor-initializer-v5.0.6.tar pingcap/tidb-monitor
   ↪ -initializer:v5.0.6
docker save -o grafana-6.0.1.tar grafana/grafana:6.0.1
docker save -o prometheus-v2.18.1.tar prom/prometheus:v2.18.1
docker save -o busybox-1.26.2.tar busybox:1.26.2
```

接下来将这些 Docker 镜像上传到服务器上，并执行 `docker load` 将这些 Docker 镜像安装到服务器上：

```
docker load -i pd-v5.0.6.tar
docker load -i tikv-v5.0.6.tar
docker load -i tidb-v5.0.6.tar
docker load -i tidb-binlog-v5.0.6.tar
docker load -i ticdc-v5.0.6.tar
docker load -i tiflash-v5.0.6.tar
```

```
docker load -i tidb-monitor-reloader-v1.0.1.tar
docker load -i tidb-monitor-initializer-v5.0.6.tar
docker load -i grafana-6.0.1.tar
docker load -i prometheus-v2.18.1.tar
docker load -i busybox-1.26.2.tar
```

3. 通过下面命令查看 Pod 状态：

```
kubectl get po -n ${namespace} -l app.kubernetes.io/instance=${
    ↪ cluster_name}
```

单个 Kubernetes 集群中可以利用 TiDB Operator 部署管理多套 TiDB 集群，重复以上步骤并将 `cluster_name` 替换成不同名字即可。不同集群既可以在相同 `namespace` 中，也可以在不同 `namespace` 中，可根据实际需求进行选择。

> **注意：**
>
> 如果要将 TiDB 集群部署到 ARM64 机器上，可以参考在 ARM64 机器上部署 TiDB 集群。

4.1.5.5.3 初始化 TiDB 集群

如果要在部署完 TiDB 集群后做一些初始化工作，参考Kubernetes 上的集群初始化配置进行配置。

> **注意：**
>
> TiDB（v4.0.2 起）默认会定期收集使用情况信息，并将这些信息分享给 PingCAP 用于改善产品。若要了解所收集的信息详情及如何禁用该行为，请参见遥测。

4.1.5.6 Kubernetes 上的集群初始化配置

本文介绍如何对 Kubernetes 上的集群进行初始化配置完成初始化账号和密码设置，以及批量自动执行 SQL 语句对数据库进行初始化。

> **注意：**

- 如果 TiDB 集群创建完以后手动修改过 root 用户的密码，初始化会失败。
- 以下功能只在 TiDB 集群创建后第一次执行起作用，执行完以后再修改不会生效。

4.1.5.6.1 配置 TidbInitializer

请参考 TidbInitializer 示例和 API 文档（示例和 API 文档请切换到当前使用的 TiDB Operator 版本）以及下面的步骤，完成 TidbInitializer CR，保存到文件 ${cluster_name}/tidb-initializer.yaml。

设置集群的命名空间和名称

在 ${cluster_name}/tidb-initializer.yaml 文件中，修改 spec.cluster.namespace 和 spec.cluster.name 字段:

```
#### ...
spec:
  # ...
  cluster:
    namespace: ${cluster_namespace}
    name: ${cluster_name}
```

初始化账号和密码设置

集群创建时默认会创建 root 账号，但是密码为空，这会带来一些安全性问题。可以通过如下步骤为 root 账号设置初始密码：

通过下面命令创建 Secret 指定 root 账号密码：

```
kubectl create secret generic tidb-secret --from-literal=root=${root_password} --namespace=${namespace}
```

如果希望能自动创建其它用户，可以在上面命令里面再加上其他用户的 username 和 password，例如：

```
kubectl create secret generic tidb-secret --from-literal=root=${root_password} --from-literal=developer=${developer_password} --namespace=${namespace}
```

该命令会创建 root 和 developer 两个用户的密码，存到 tidb-secret 的 Secret 里面。并且创建的普通用户 developer 默认只有 USAGE 权限，其他权限请在 initSql 中设置。

在 ${cluster_name}/tidb-initializer.yaml 中设置 passwordSecret: tidb-secret。

4.1.5.6.2 设置允许访问 TiDB 的主机

在 `${cluster_name}/tidb-initializer.yaml` 中设置 `permitHost: ${mysql_client_host_name}` 配置项来设置允许访问 TiDB 的主机 **host_name**。如果不设置，则允许所有主机访问。详情请参考 MySQL GRANT host name。

4.1.5.6.3 批量执行初始化 SQL 语句

集群在初始化过程还可以自动执行 `initSql` 中的 SQL 语句用于初始化，该功能可以用于默认给集群创建一些 database 或者 table，并且执行一些用户权限管理类的操作。例如如下设置会在集群创建完成后自动创建名为 `app` 的 database，并且赋予 `developer` 账号对 `app` 的所有管理权限：

```
spec:
...
initSql: |-
    CREATE DATABASE app;
    GRANT ALL PRIVILEGES ON app.* TO 'developer'@'%';
```

> **注意：**
>
> 目前没有对 initSql 做校验，尽管也可以在 initSql 里面创建账户和设置密码，但这种方式会将密码以明文形式存到 initializer Job 对象上，不建议这么做。

4.1.5.6.4 执行初始化

```
kubectl apply -f ${cluster_name}/tidb-initializer.yaml --namespace=${namespace}
```

以上命令会自动创建一个初始化的 Job，该 Job 会尝试利用提供的 secret 给 root 账号创建初始密码，并且创建其它账号和密码（如果指定了的话）。初始化完成后 Pod 状态会变成 Completed，之后通过 MySQL 客户端登录时需要指定这里设置的密码。

如果服务器没有外网，需要在有外网的机器上将集群初始化用到的 Docker 镜像下载下来并上传到服务器上，然后使用 `docker load` 将 Docker 镜像安装到服务器上。

初始化一套 TiDB 集群会用到下面这些 Docker 镜像：

```
tnir/mysqlclient:latest
```

接下来通过下面的命令将所有这些镜像下载下来：

```
docker pull tnir/mysqlclient:latest
docker save -o mysqlclient-latest.tar tnir/mysqlclient:latest
```

接下来将这些 Docker 镜像上传到服务器上，并执行 `docker load` 将这些 Docker 镜像安装到服务器上：

```
docker load -i mysqlclient-latest.tar
```

4.1.5.7 访问 TiDB 集群

Service 可以根据场景配置不同的类型，比如 `ClusterIP`、`NodePort`、`LoadBalancer` 等，对于不同的类型可以有不同的访问方式。

可以通过如下命令获取 TiDB Service 信息：

```
kubectl get svc ${serviceName} -n ${namespace}
```

示例：

```
#### kubectl get svc basic-tidb -n default
NAME        TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)
    ↪ AGE
basic-tidb  NodePort  10.233.6.240  <none>       4000:32498/TCP,10080:30171/
    ↪ TCP 61d
```

上述示例描述了 `default` namespace 下 `basic-tidb` 服务的信息，类型为 `NodePort`，ClusterIP 为 `10.233.6.240`，ServicePort 为 `4000` 和 `10080`，对应的 NodePort 分别为 `32498` 和 `30171`。

> 注意：
>
> MySQL 8.0 默认认证插件从 `mysql_native_password` 更新为 `caching_sha2_password`，因此如果使用 MySQL 8.0 客户端访问 TiDB 服务（TiDB 版本 < v4.0.7），并且用户账户有配置密码，需要显示指定 `--default-auth=mysql_native_password` 参数。

4.1.5.7.1 ClusterIP

`ClusterIP` 是通过集群的内部 IP 暴露服务，选择该类型的服务时，只能在集群内部访问，可以通过如下方式访问：

- ClusterIP + ServicePort
- Service 域名 (`${serviceName}.${namespace}`) + ServicePort

4.1.5.7.2 NodePort

在没有 LoadBalancer 时，可选择通过 NodePort 暴露。NodePort 是通过节点的 IP 和静态端口暴露服务。通过请求 `NodeIP + NodePort`，可以从集群的外部访问一个 NodePort 服务。

查看 Service 分配的 Node Port，可通过获取 TiDB 的 Service 对象来获知：

```
kubectl -n ${namespace} get svc ${cluster_name}-tidb -ojsonpath="{.spec.
    ↪ ports[?(@.name=='mysql-client')].nodePort}{'\n'}"
```

查看可通过哪些节点的 IP 访问 TiDB 服务，有两种情况：

- `externalTrafficPolicy` 为 `Cluster` 时，所有节点 IP 均可
- `externalTrafficPolicy` 为 `Local` 时，可通过以下命令获取指定集群的 TiDB 实例所在的节点

```
kubectl -n ${namespace} get pods -l "app.kubernetes.io/component=tidb,
    ↪ app.kubernetes.io/instance=${cluster_name}" -ojsonpath="{range .
    ↪ items[*]}{.spec.nodeName}{'\n'}{end}"
```

4.1.5.7.3 LoadBalancer

若运行在有 LoadBalancer 的环境，比如 GCP/AWS 平台，建议使用云平台的 LoadBalancer 特性。

参考EKS、GKE 和ACK 文档，通过 LoadBalancer 访问 TiDB 服务。

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

4.1.6 在 ARM64 机器上部署 TiDB 集群

本文档介绍如何在 ARM64 机器上部署 TiDB 集群。

4.1.6.1 前置条件

- 在 ARM64 机器上已经部署了 Kubernetes。如果尚未部署，请参阅部署 Kubernetes 集群。

4.1.6.2 部署 TiDB Operator

在 ARM64 机器上部署 TiDB Operator 的步骤与在 Kubernetes 上部署 TiDB Operator 的步骤相同。唯一区别是，在自定义部署 TiDB Operator 这一步，当获取到 `tidb-operator` chart 中的 `value.yaml` 文件后，你需要修改文件中的 `operatorImage` 与 `tidbBackupManagerImage` 字段为 ARM64 版本镜像。例如：

```
### ...
operatorImage: pingcap/tidb-operator-arm64:v1.2.4
### ...
tidbBackupManagerImage: pingcap/tidb-backup-manager-arm64:v1.2.4
### ...
```

4.1.6.3 部署 TiDB 集群

在 ARM64 机器上部署 TiDB 集群的步骤与在标准 Kubernetes 上部署 TiDB 集群 的步骤相同。唯一区别是，你需要将 TidbCluster 定义文件中相关组件的镜像设置为 ARM64 版本。例如：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${cluster_name}
  namespace: ${cluster_namespace}
spec:
  version: "v5.2.1"
  # ...
  helper:
    image: busybox:1.33.0
  # ...
  pd:
    baseImage: pingcap/pd-arm64
    # ...
  tidb:
    baseImage: pingcap/tidb-arm64
    # ...
  tikv:
    baseImage: pingcap/tikv-arm64
    # ...
  pump:
    baseImage: pingcap/tidb-binlog-arm64
    # ...
  ticdc:
    baseImage: pingcap/ticdc-arm64
    # ...
  tiflash:
    baseImage: pingcap/tiflash-arm64
    # ...
```

4.1.6.4 初始化 TiDB 集群

在 ARM64 机器上初始化 TiDB 集群的步骤与在 Kubernetes 上的初始化 TiDB 集群的步骤 相同。唯一区别是，你需要将 TidbInitializer 定义文件中的 spec.image 字段设置为 ARM64 版本镜像。例如：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbInitializer
metadata:
  name: ${initializer_name}
  namespace: ${cluster_namespace}
spec:
  image: kanshiori/mysqlclient-arm64
  # ...
```

4.1.6.5 部署 TiDB 集群监控

在 ARM64 机器上部署 TiDB 集群监控的步骤与TiDB 集群的监控与告警 的步骤相同。唯一区别是，你需要将 TidbMonitor 定义文件中的 spec.initializer.baseImage 与 spec.reloader.baseImage 字段设置为 ARM64 版本镜像。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ${monitor_name}
spec:
  # ...
  initializer:
    baseImage: pingcap/tidb-monitor-initializer-arm64
    version: v5.2.1
  reloader:
    baseImage: pingcap/tidb-monitor-reloader-arm64
    version: v1.0.1
  # ...
```

4.2 为已有 TiDB 集群部署异构集群

本文档介绍如何为已有的 TiDB 集群部署一个异构集群。

4.2.1 前置条件

- 已经存在一个 TiDB 集群，可以参考在标准 Kubernetes 上部署 TiDB 集群进行部署。

4.2.2 部署异构集群

4.2.2.1 什么是异构集群

异构集群是给已经存在的 TiDB 集群创建差异化的实例节点，比如创建不同配置不同 Label 的 TiKV 集群用于热点调度或者创建不同配置的 TiDB 集群分别用于 TP 和 AP 查询。

4.2.2.2 创建一个异构集群

将如下配置存为 `cluster.yaml` 文件，并替换 `${heterogeneous_cluster_name}` 为自己想命名的异构集群名字，`${origin_cluster_name}` 替换为想要加入的已有集群名称:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${heterogeneous_cluster_name}
spec:
  configUpdateStrategy: RollingUpdate
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  discovery: {}
  cluster:
    name: ${origin_cluster_name}
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
    # if storageClassName is not set, the default Storage Class of the
       ↪ Kubernetes cluster will be used
    # storageClassName: local-storage
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    service:
      type: ClusterIP
    config: {}
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 1
    replicas: 1
    storageClaims:
      - resources:
          requests:
            storage: 1Gi
        storageClassName: standard
```

执行以下命令创建集群：

```
kubectl create -f cluster.yaml -n ${namespace}
```

异构集群除了使用 spec.cluster.name 字段加入到目标集群，其它字段和正常的 TiDB 集群一样。

4.2.2.3 部署集群监控

将如下配置存为 tidbmonitor.yaml 文件，并替换 ${origin_cluster_name} 为想要加入的集群名称，${heterogeneous_cluster_name} 替换为异构集群名称：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: heterogeneous
spec:
  clusters:
    - name: ${origin_cluster_name}
    - name: ${heterogeneous_cluster_name}
  prometheus:
    baseImage: prom/prometheus
    version: v2.11.1
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

执行以下命令创建集群：

```
kubectl create -f tidbmonitor.yaml -n ${namespace}
```

4.2.3 部署 TLS 异构集群

开启异构集群 TLS 需要显示声明，需要创建新的 Secret 证书文件，使用和目标集群相同的 CA (Certification Authority) 颁发。如果使用 cert-manager 方式，需要使用和目标集群相同的 Issuer 来创建 Certificate。

为异构集群创建证书的详细步骤，可参考以下文档：

- 为 TiDB 组件间开启 TLS
- 为 MySQL 客户端开启 TLS

4.2.3.1 创建一个异构 TLS 集群

将如下配置存为 cluster.yaml 文件，并替换 ${heterogeneous_cluster_name} 为自己想命名的异构集群名字，${origin_cluster_name} 替换为想要加入的已有集群名称:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${heterogeneous_cluster_name}
spec:
  tlsCluster:
    enabled: true
  configUpdateStrategy: RollingUpdate
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  discovery: {}
  cluster:
    name: ${origin_cluster_name}
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
    # if storageClassName is not set, the default Storage Class of the
       ↪ Kubernetes cluster will be used
    # storageClassName: local-storage
    requests:
      storage: "1Gi"
    config:
      storage:
        # In basic examples, we set this to avoid using too much storage.
        reserve-space: "0MB"
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    service:
      type: ClusterIP
    config: {}
    tlsClient:
      enabled: true
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 1
    replicas: 1
    storageClaims:
      - resources:
          requests:
```

```
          storage: 1Gi
      storageClassName: standard
```

spec.tlsCluster.enabled 表示组件间是否开启 TLS，spec.tidb.tlsClient.enabled 表示 MySQL 客户端是否开启 TLS。

执行以下命令创建开启 TLS 的异构集群：

```
kubectl create -f cluster.yaml -n ${namespace}
```

详细的异构 TLS 集群配置示例，请参阅 'heterogeneous-tls'。

4.3 在 Kubernetes 上部署 TiFlash

本文介绍如何在 Kubernetes 上部署 TiFlash。

4.3.1 前置条件

- TiDB Operator 部署完成。

4.3.2 全新部署 TiDB 集群同时部署 TiFlash

参考在标准 Kubernetes 上部署 TiDB 集群进行部署。

4.3.3 在现有 TiDB 集群上新增 TiFlash 组件

编辑 TidbCluster Custom Resource：

```
kubectl edit tc ${cluster_name} -n ${namespace}
```

按照如下示例增加 TiFlash 配置：

```
spec:
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 3
    replicas: 1
    storageClaims:
    - resources:
        requests:
          storage: 100Gi
      storageClassName: local-storage
```

其他参数可以参考集群配置文档进行配置。

TiFlash 支持挂载多个 PV，如果要为 TiFlash 配置多个 PV，可以在 `tiflash.storageClaims` 下面配置多项，每一项可以分别配置 `storage reqeust` 和 `storageClassName`，例如：

```
tiflash:
  baseImage: pingcap/tiflash
  maxFailoverCount: 3
  replicas: 1
  storageClaims:
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
```

警告：

由于 TiDB Operator 会按照 `storageClaims` 列表中的配置按顺序自动挂载 PV，如果需要为 TiFlash 增加磁盘，请确保只在列表原有配置最后添加，并且不能修改列表中原有配置的顺序。

TiDB Operator 通过创建 StatefulSet 管理 TiFlash，由于 `StatefulSet` 创建后不支持修改 `volumeClaimTemplates`，因此直接更新 `storageClaims` 添加磁盘不会为 Pod 挂载上额外的 PV，解决方案有下面两种：

- 第一次部署 TiFlash 集群就规划好使用几个 PV，配置好 `storageClaims`。
- 如果确实要新增 PV，配置好 `storageClaims` 后，需要手动删除 TiFlash StatefulSet (`kubectl delete sts -n ${namespace} ${cluster_name}-tiflash`)，等待 TiDB Operator 重新创建。

新增部署 TiFlash 需要 PD 配置 `replication.enable-placement-rules: true`，通过上述步骤在 TidbCluster 中增加 TiFlash 配置后，TiDB Operator 会自动为 PD 配置 `replication.enable-placement-rules: true`。

如果服务器没有外网，请参考部署 TiDB 集群在有外网的机器上将用到的 Docker 镜像下载下来并上传到服务器上。

4.3.4 移除 TiFlash

1. 调整同步到 TiFlash 集群中的数据表的副本数。

   需要将集群中所有同步到 TiFlash 的数据表的副本数都设置为 0，才能完全移除 TiFlash。

   1. 参考访问 TiDB 集群的步骤连接到 TiDB 服务。
   2. 使用以下命令，调整同步到 TiFlash 集群中的数据表的副本数：

   ```
   alter table <db_name>.<table_name> set tiflash replica 0;
   ```

2. 等待相关表的 TiFlash 副本被删除。

   连接到 TiDB 服务，执行如下命令，查不到相关表的同步信息时即为副本被删除：

   ```
   SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA =
       ↪ '<db_name>' and TABLE_NAME = '<table_name>';
   ```

3. 执行以下命令修改 spec.tiflash.replicas 为 0 来移除 TiFlash Pod。

   ```
   kubectl edit tidbcluster ${cluster_name} -n ${namespace}
   ```

4. 检查 TiFlash Pod 和 TiFlash 节点 store 状态。

   首先执行以下命令检查 TiFlash Pod 是否被成功删除：

   ```
   kubectl get pod -n ${namespace} -l app.kubernetes.io/component=tiflash,
       ↪ app.kubernetes.io/instance=${cluster_name}
   ```

   如果输出为空，则表示 TiFlash 集群的 Pod 已经被成功删除。

   使用以下命令检查 TiFlash 节点 store 状态是否为 Tombstone:

   ```
   kubectl get tidbcluster ${cluster_name} -n ${namespace} -o yaml
   ```

   输出结果中的 status.tiflash 字段值类似下方实例。

   ```
   tiflash:
       ...
       tombstoneStores:
       "88":
           id: "88"
           ip: basic-tiflash-0.basic-tiflash-peer.default.svc
           lastHeartbeatTime: "2020-12-31T04:42:12Z"
           lastTransitionTime: null
           leaderCount: 0
           podName: basic-tiflash-0
           state: Tombstone
       "89":
   ```

```
      id: "89"
      ip: basic-tiflash-1.basic-tiflash-peer.default.svc
      lastHeartbeatTime: "2020-12-31T04:41:50Z"
      lastTransitionTime: null
      leaderCount: 0
      podName: basic-tiflash-1
      state: Tombstone
```

只有 TiFlash 集群的所有 Pod 已经被成功删除并且所有 TiFlash 节点 store 状态都变为 Tombstone 后，才能进行下一步操作。

5. 删除 TiFlash StatefulSet。

   使用以下命令修改 TiDB Cluster CR，删除 `spec.tiflash` 字段。

```
kubectl edit tidbcluster ${cluster_name} -n ${namespace}
```

   使用以下命令删除 TiFlash StatefulSet：

```
kubectl delete statefulsets -n ${namespace} -l app.kubernetes.io/
    ↪ component=tiflash,app.kubernetes.io/instance=${cluster_name}
```

   执行以下命令检查是否成功删除 TiFlash 集群的 StatefulSet：

```
kubectl get sts -n ${namespace} -l app.kubernetes.io/component=tiflash,
    ↪ app.kubernetes.io/instance=${cluster_name}
```

   如果输出为空，则表示 TiFlash 集群的 StatefulSet 已经被成功删除。

6. (可选项) 删除 PVC 和 PV。

   如果确认 TiFlash 中的数据不会被使用，想要删除数据，需要严格按照以下操作步骤来删除 TiFlash 中的数据。

   1. 删除 PV 对应的 PVC 对象

```
kubectl delete pvc -n ${namespace} -l app.kubernetes.io/component=
    ↪ tiflash,app.kubernetes.io/instance=${cluster_name}
```

   2. PV 保留策略是 Retain 时，删除 PVC 对象后对应的 PV 仍将保留。如果想要删除 PV，可以设置 PV 的保留策略为 Delete，PV 会被自动删除并回收。

```
kubectl patch pv ${pv_name} -p '{"spec":{"
    ↪ persistentVolumeReclaimPolicy":"Delete"}}'
```

      其中 `${pv_name}` 表示 TiFlash 集群 PV 的名称，可以执行以下命令查看：

```
kubectl get pv -l app.kubernetes.io/component=tiflash,app.
    ↪ kubernetes.io/instance=${cluster_name}
```

4.3.5 不同版本配置注意事项

从 TiDB Operator v1.1.5 版本开始，`spec.tiflash.config.config.flash.service_addr` 的默认配置从 `${clusterName}-tiflash-POD_NUM.${clusterName}-tiflash-peer.${namespace}.svc:3930` 修改为 `0.0.0.0:3930`，而 TiFlash 从 v4.0.5 开始需要配置 `spec.tiflash.config.config.flash.service_addr` 为 `0.0.0.0:3930`，因此针对不同 TiFlash 和 TiDB Operator 版本，需要注意以下配置：

- 如果 TiDB Operator 版本 <= v1.1.4
  - 如果 TiFlash 版本 <= v4.0.4，不需要手动配置 `spec.tiflash.config.config.flash.service_addr`。
  - 如果 TiFlash 版本 >= v4.0.5，需要在 TidbCluster CR 中设置 `spec.tiflash.config.config.flash.service_addr` 为 `0.0.0.0:3930`。
- 如果 TiDB Operator 版本 >= v1.1.5
  - 如果 TiFlash 版本 <= v4.0.4，需要在 TidbCluster CR 中设置 `spec.tiflash.config.config.flash.service_addr` 为 `${clusterName}-tiflash-POD_NUM.${clusterName}-tiflash-peer.${namespace}.svc:3930`。其中，`${clusterName}` 和 `${namespace}` 需要根据实际情况替换。
  - 如果 TiFlash 版本 >= v4.0.5，不需要手动配置 `spec.tiflash.config.config.flash.service_addr`。
  - 如果从小于等于 v4.0.4 的 TiFlash 版本升级到大于等于 v4.0.5 TiFlash 版本，需要删除 TidbCluster CR 中 `spec.tiflash.config.config.flash.service_addr` 的配置。

4.4 在 Kubernetes 上部署 TiCDC

TiCDC 是一款 TiDB 增量数据同步工具，本文介绍如何使用 TiDB Operator 在 Kubernetes 上部署 TiCDC。

4.4.1 前置条件

- TiDB Operator 部署完成。

4.4.2 全新部署 TiDB 集群同时部署 TiCDC

参考在标准 Kubernetes 上部署 TiDB 集群进行部署。

4.4.3 在现有 TiDB 集群上新增 TiCDC 组件

1. 编辑 TidbCluster Custom Resource：

```
kubectl edit tc ${cluster_name} -n ${namespace}
```

2. 按照如下示例增加 TiCDC 配置：

```
spec:
  ticdc:
    baseImage: pingcap/ticdc
    replicas: 3
```

3. 部署完成后，通过 kubectl exec 进入任意一个 TiCDC Pod 进行操作。

```
kubectl exec -it ${pod_name} -n ${namespace} -- sh
```

4. 然后通过 cdc cli 进行管理集群和同步任务。

```
/cdc cli capture list --pd=http://${cluster_name}-pd:2379
```

```
[
  {
    "id": "3ed24f6c-22cf-446f-9fe0-bf4a66d00f5b",
    "is-owner": false,
    "address": "${cluster_name}-ticdc-2.${cluster_name}-ticdc-peer.${
       ↪ namespace}.svc:8301"
  },
  {
    "id": "60e98ed7-cd49-45f4-b5ae-d3b85ba3cd96",
    "is-owner": false,
    "address": "${cluster_name}-ticdc-0.${cluster_name}-ticdc-peer.${
       ↪ namespace}.svc:8301"
  },
  {
    "id": "dc3592c0-dace-42a0-8afc-fb8506e8271c",
    "is-owner": true,
    "address": "${cluster_name}-ticdc-1.${cluster_name}-ticdc-peer.${
       ↪ namespace}.svc:8301"
  }
]
```

TiCDC 从 v4.0.3 版本开始支持 TLS，TiDB Operator v1.1.3 版本同步支持 TiCDC 开启 TLS 功能。

如果在创建 TiDB 集群时开启了 TLS，使用 cdc cli 请携带 TLS 证书相关参数：

```
/cdc cli capture list --pd=https://${cluster_name}-pd:2379 --ca=/var/
   ↪ lib/cluster-client-tls/ca.crt --cert=/var/lib/cluster-client-tls/
   ↪ tls.crt --key=/var/lib/cluster-client-tls/tls.key
```

如果服务器没有外网，请参考部署 TiDB 集群 在有外网的机器上将用到的 Docker 镜像下载下来并上传到服务器上。

4.5 部署 TiDB Binlog

本文档介绍如何在 Kubernetes 上部署 TiDB 集群的 TiDB Binlog。

4.5.1 部署准备

- 部署 TiDB Operator；
- 安装 Helm 并配置 PingCAP 官方 chart 仓库。

4.5.2 部署 TiDB 集群的 TiDB Binlog

默认情况下，TiDB Binlog 在 TiDB 集群中处于禁用状态。若要创建一个启用 TiDB Binlog 的 TiDB 集群，或在现有 TiDB 集群中启用 TiDB Binlog，可根据以下步骤进行操作。

4.5.2.1 部署 Pump

可以修改 TidbCluster CR，添加 Pump 相关配置，示例如下：

```
spec
  ...
  pump:
    baseImage: pingcap/tidb-binlog
    version: v5.0.6
    replicas: 1
    storageClassName: local-storage
    requests:
      storage: 30Gi
    schedulerName: default-scheduler
    config:
      addr: 0.0.0.0:8250
      gc: 7
      heartbeat-interval: 2
```

自 v1.1.6 版本起支持透传 TOML 配置给组件:

```
spec
  ...
  pump:
    baseImage: pingcap/tidb-binlog
    version: v5.0.6
    replicas: 1
    storageClassName: local-storage
    requests:
      storage: 30Gi
    schedulerName: default-scheduler
```

```
  config: |
    addr = "0.0.0.0:8250"
    gc = 7
    heartbeat-interval = 2
```

按照集群实际情况修改 version、replicas、storageClassName、requests.storage 等配置。

如果在生产环境中开启 TiDB Binlog，建议为 TiDB 与 Pump 组件设置亲和性和反亲和性。如果在内网测试环境中尝试使用开启 TiDB Binlog，可以跳过此步。

默认情况下，TiDB 和 Pump 的 affinity 亲和性设置为 {}。由于目前 Pump 组件与 TiDB 组件默认并非一一对应，当启用 TiDB Binlog 时，如果 Pump 与 TiDB 组件分开部署并出现网络隔离，而且 TiDB 组件还开启了 ignore-error，则会导致 TiDB 丢失 Binlog。推荐通过亲和性特性将 TiDB 组件与 Pump 部署在同一台 Node 上，同时通过反亲和性特性将 Pump 分散在不同的 Node 上，每台 Node 上至多仅需一个 Pump 实例。

- 将 spec.tidb.affinity 按照如下设置：

```
spec:
  tidb:
    affinity:
      podAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: "app.kubernetes.io/component"
                operator: In
                values:
                - "pump"
              - key: "app.kubernetes.io/managed-by"
                operator: In
                values:
                - "tidb-operator"
              - key: "app.kubernetes.io/name"
                operator: In
                values:
                - "tidb-cluster"
              - key: "app.kubernetes.io/instance"
                operator: In
                values:
                - ${cluster_name}
            topologyKey: kubernetes.io/hostname
```

- 将 spec.pump.affinity 按照如下设置：

```
spec:
  pump:
    affinity:
      podAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: "app.kubernetes.io/component"
                operator: In
                values:
                - "tidb"
              - key: "app.kubernetes.io/managed-by"
                operator: In
                values:
                - "tidb-operator"
              - key: "app.kubernetes.io/name"
                operator: In
                values:
                - "tidb-cluster"
              - key: "app.kubernetes.io/instance"
                operator: In
                values:
                - ${cluster_name}
            topologyKey: kubernetes.io/hostname
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: "app.kubernetes.io/component"
                operator: In
                values:
                - "pump"
              - key: "app.kubernetes.io/managed-by"
                operator: In
                values:
                - "tidb-operator"
              - key: "app.kubernetes.io/name"
                operator: In
                values:
```

```
            - "tidb-cluster"
          - key: "app.kubernetes.io/instance"
            operator: In
            values:
            - ${cluster_name}
        topologyKey: kubernetes.io/hostname
```

注意：

如果更新了 TiDB 组件的亲和性配置，将引起 TiDB 组件滚动更新。

4.5.2.2 部署 Drainer

可以通过 `tidb-drainer` Helm chart 来为 TiDB 集群部署多个 drainer，示例如下：

1. 确保 PingCAP Helm 库是最新的：

```
helm repo update
```

```
helm search repo tidb-drainer -l
```

2. 获取默认的 `values.yaml` 文件以方便自定义：

```
helm inspect values pingcap/tidb-drainer --version=${chart_version} >
    ↪ values.yaml
```

3. 修改 `values.yaml` 文件以指定源 TiDB 集群和 drainer 的下游数据库。示例如下：

```
clusterName: example-tidb
clusterVersion: v5.0.6
baseImage: pingcap/tidb-binlog
storageClassName: local-storage
storage: 10Gi
initialCommitTs: "-1"
config: |
  detect-interval = 10
  [syncer]
  worker-count = 16
  txn-batch = 20
  disable-dispatch = false
  ignore-schemas = "INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql"
  safe-mode = false
```

```
  db-type = "tidb"
  [syncer.to]
  host = "downstream-tidb"
  user = "root"
  password = ""
  port = 4000
```

clusterName 和 clusterVersion 必须匹配所需的源 TiDB 集群。

initialCommitTs 为 drainer 没有 checkpoint 时数据同步的起始 commit timestamp。该参数值必须以 string 类型配置，如 "424364429251444742"。

有关完整的配置详细信息，请参阅Kubernetes 上的 TiDB Binlog Drainer 配置。

4. 部署 Drainer：

```
helm install ${release_name} pingcap/tidb-drainer --namespace=${
    ↪ namespace} --version=${chart_version} -f values.yaml
```

如果服务器没有外网，请参考部署 TiDB 集群 在有外网的机器上将用到的 Docker 镜像下载下来并上传到服务器上。

> 注意：
> 该 chart 必须与源 TiDB 集群安装在相同的命名空间中。

4.5.3 开启 TLS

4.5.3.1 为 TiDB 组件间开启 TLS

如果要为 TiDB 集群及 TiDB Binlog 开启 TLS，请参考为 TiDB 组件间开启 TLS 进行配置。

创建 secret 并启动包含 Pump 的 TiDB 集群后，修改 values.yaml 将 tlsCluster.enabled 设置为 true，并配置相应的 certAllowedCN：

```
...
tlsCluster:
  enabled: true
  # certAllowedCN:
  #  - TiDB
...
```

4.5.3.2 为 Drainer 和下游数据库间开启 TLS

如果 tidb-drainer 的写入下游设置为 mysql/tidb，并且希望为 drainer 和下游数据库间开启 TLS，可以参考下面步骤进行配置。

首先我们需要创建一个包含下游数据库 TLS 信息的 secret，创建方式如下：

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

默认情况下，tidb-drainer 会将 checkpoint 保存到下游数据库中，所以仅需配置 tlsSyncer.tlsClientSecretName 并配置相应的 certAllowedCN 即可。

```
tlsSyncer:
  tlsClientSecretName: ${downstream_database_secret_name}
  # certAllowedCN:
  #  - TiDB
```

如果需要将 tidb-drainer 的 checkpoint 保存到其他开启 **TLS** 的数据库，需要创建一个包含 checkpoint 数据库的 TLS 信息的 secret，创建方式为：

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

修改 values.yaml 将 tlsSyncer.checkpoint.tlsClientSecretName 设置为 ${
↪ checkpoint_tidb_client_secret}，并配置相应的 certAllowedCN：

```
...
tlsSyncer: {}
  tlsClientSecretName: ${downstream_database_secret_name}
  # certAllowedCN:
  #  - TiDB
  checkpoint:
    tlsClientSecretName: ${checkpoint_tidb_client_secret}
    # certAllowedCN:
    #  - TiDB
...
```

4.5.4 缩容/移除 Pump/Drainer 节点

如需详细了解如何维护 TiDB Binlog 集群节点状态信息，可以参考 Pump/Drainer 的启动、退出流程。

如果需要完整移除 TiDB Binlog 组件，最好是先移除 Pump 节点，再移除 Drainer 节点。

如果需要移除的 TiDB Binlog 组件开启了 TLS，则需要先将下述文件写入 binlog.
↪ yaml，并使用 kubectl apply -f binlog.yaml 启动一个挂载了 TLS 文件和 binlogctl 工具的 Pod。

```
apiVersion: v1
```

```
kind: Pod
metadata:
  name: binlogctl
spec:
  containers:
  - name: binlogctl
    image: pingcap/tidb-binlog:${tidb_version}
    command: ['/bin/sh']
    stdin: true
    stdinOnce: true
    tty: true
    volumeMounts:
      - name: binlog-tls
        mountPath: /etc/binlog-tls
  volumes:
    - name: binlog-tls
      secret:
        secretName: ${cluster_name}-cluster-client-secret
```

4.5.4.1 缩容 Pump 节点

缩容 Pump 需要先将单个 Pump 节点从集群中下线，然后运行 `kubectl edit tc ${cluster_name} -n ${namespace}` 命令将 Pump 对应的 replica 数量减 1，并对每个节点重复上述步骤。具体操作步骤如下：

1. 下线 Pump 节点：

   假设现在有 3 个 Pump 节点，我们需要下线第 3 个 Pump 节点，将 `${ordinal_id}` 替换成 2，操作方式如下（`${tidb_version}` 为当前 TiDB 的版本）。

   如果 Pump 没有开启 TLS，使用下述指令新建 Pod 下线 Pump。

   ```
   kubectl run offline-pump-${ordinal_id} --image=pingcap/tidb-binlog:${
       ↪ tidb_version} --namespace=${namespace} --restart=OnFailure -- /
       ↪ binlogctl -pd-urls=http://${cluster_name}-pd:2379 -cmd offline-
       ↪ pump -node-id ${cluster_name}-pump-${ordinal_id}:8250
   ```

   如果 Pump 开启了 TLS，通过下述指令使用前面开启的 Pod 来下线 Pump。

   ```
   kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls "https://
       ↪ ${cluster_name}-pd:2379" -cmd offline-pump -node-id ${
       ↪ cluster_name}-pump-${ordinal_id}:8250 -ssl-ca "/etc/binlog-tls/ca
       ↪ .crt" -ssl-cert "/etc/binlog-tls/tls.crt" -ssl-key "/etc/binlog-
       ↪ tls/tls.key"
   ```

   然后查看 Pump 的日志输出，输出 `pump offline, please delete my pod` 后即可确认该节点已经成功下线。

```
kubectl logs -f -n ${namespace} ${release_name}-pump-${ordinal_id}
```

2. 删除对应的 Pump Pod：

   运行 `kubectl edit tc ${cluster_name} -n ${namespace}` 修改文件中 `spec.pump.replicas` 为 2，然后等待 Pump Pod 自动下线被删除。

3. (可选项) 强制下线 Pump

   如果在下线 Pump 节点时遇到下线失败的情况，即执行下线操作后仍未看到 Pump pod 输出可以删除 pod 的日志，可以先进行步骤 2 调小 `replicas`，等待 Pump Pod 被完全删除后，标注 Pump 状态为 offline。

   没有开启 TLS 时，使用下述指令标注状态为 offline。

```
kubectl run update-pump-${ordinal_id} --image=pingcap/tidb-binlog:${
    ↪ tidb_version} --namespace=${namespace} --restart=OnFailure -- /
    ↪ binlogctl -pd-urls=http://${cluster_name}-pd:2379 -cmd update-
    ↪ pump -node-id ${cluster_name}-pump-${ordinal_id}:8250 --state
    ↪ offline
```

   如果开启了 TLS，通过下述指令使用前面开启的 pod 来标注状态为 offline。

```
kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls=https://$
    ↪ {cluster_name}-pd:2379 -cmd update-pump -node-id ${cluster_name}-
    ↪ pump-${ordinal_id}:8250 --state offline -ssl-ca "/etc/binlog-tls/
    ↪ ca.crt" -ssl-cert "/etc/binlog-tls/tls.crt" -ssl-key "/etc/binlog
    ↪ -tls/tls.key"
```

4.5.4.2 完全移除 Pump 节点

> **注意：**
>
> 执行如下步骤之前，集群内需要至少存在一个 Pump 节点。如果此时 Pump 节点已经缩容到 0，需要先至少扩容到 1，再进行下面的移除操作。如果需要扩容至 1，使用命令 `kubectl edit tc ${tidb-cluster} -n ${namespace}`，修改 `spec.pump.replicas` 为 1 即可。

1. 移除 Pump 节点前，必须首先需要执行 `kubectl edit tc ${cluster_name} -n ${namespace}` 设置其中的 `spec.tidb.binlogEnabled` 为 `false`，等待 TiDB Pod 完成重启更新后再移除 Pump 节点。如果直接移除 Pump 节点会导致 TiDB 没有可以写入的 Pump 而无法使用。
2. 参考缩容 Pump 节点步骤缩容 Pump 到 0。

3. kubectl edit tc ${cluster_name} -n ${namespace} 将 spec.pump 部分配置项全部删除。
4. kubectl delete sts ${cluster_name}-pump -n ${namespace} 删除 Pump StatefulSet 资源。
5. 通过 kubectl get pvc -n ${namespace} -l app.kubernetes.io/component=pump 查看 Pump 集群使用过的 PVC，随后使用 kubectl delete pvc -l app.kubernetes.io/component=pump -n ${namespace} 指令删除 Pump 的所有 PVC 资源。

4.5.4.3 移除 Drainer 节点

1. 下线 Drainer 节点：

   使用下述指令下线 Drainer 节点，${drainer_node_id} 为需要下线的 Drainer 的 node ID。如果在 Helm 的 values.yaml 中配置了 drainerName 选项，则 ${drainer_node_id} 为 ${drainer_name}-0，否则 ${drainer_node_id} 为 ${cluster_name}-${release_name}-drainer-0。

   如果 Drainer 没有开启 TLS，使用下述指令新建 pod 下线 Drainer。

```
kubectl run offline-drainer-0 --image=pingcap/tidb-binlog:${tidb_version} --namespace=${namespace} --restart=OnFailure -- /binlogctl -pd-urls=http://${cluster_name}-pd:2379 -cmd offline-drainer -node-id ${drainer_node_id}:8249
```

   如果 Drainer 开启了 TLS，通过下述指令使用前面开启的 pod 来下线 Drainer。

```
kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls "https://${cluster_name}-pd:2379" -cmd offline-drainer -node-id ${drainer_node_id}:8249 -ssl-ca "/etc/binlog-tls/ca.crt" -ssl-cert "/etc/binlog-tls/tls.crt" -ssl-key "/etc/binlog-tls/tls.key"
```

   然后查看 Drainer 的日志输出，输出 drainer offline, please delete my pod 后即可确认该节点已经成功下线。

```
kubectl logs -f -n ${namespace} ${drainer_node_id}
```

2. 删除对应的 Drainer Pod：

   运行 helm uninstall ${release_name} -n ${namespace} 指令即可删除 Drainer Pod。

   如果不再使用 Drainer，使用 kubectl delete pvc data-${drainer_node_id} -n ${namespace} 指令删除该 Drainer 的 PVC 资源。

3. (可选项) 强制下线 Drainer

   如果在下线 Drainer 节点时遇到下线失败的情况，即执行下线操作后仍未看到 Drainer pod 输出可以删除 pod 的日志，可以先进行步骤 2 删除 Drainer Pod 后，再运行下述指令标注 Drainer 状态为 offline：

没有开启 TLS 时，使用下述指令标注状态为 offline。

```
kubectl run update-drainer-${ordinal_id} --image=pingcap/tidb-binlog:${
    ↪ tidb_version} --namespace=${namespace} --restart=OnFailure -- /
    ↪ binlogctl -pd-urls=http://${cluster_name}-pd:2379 -cmd update-
    ↪ drainer -node-id ${drainer_node_id}:8249 --state offline
```

如果开启了 TLS，通过下述指令使用前面开启的 pod 来下线 Drainer。

```
kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls=https://$
    ↪ {cluster_name}-pd:2379 -cmd update-drainer -node-id ${
    ↪ drainer_node_id}:8249 --state offline -ssl-ca "/etc/binlog-tls/ca
    ↪ .crt" -ssl-cert "/etc/binlog-tls/tls.crt" -ssl-key "/etc/binlog-
    ↪ tls/tls.key"
```

4.6 部署多套 TiDB Operator 分别管理不同的 TiDB 集群

本文介绍如何部署多套 TiDB Operator，分别管理不同的 TiDB 集群。

注意：

- 目前仅支持部署多套 tidb-controller-manager 和 tidb-scheduler，不支持部署多套 AdvancedStatefulSet controller 和 AdmissionWebhook。
- 如果部署了多套 TiDB Operator，有的开启了`Advanced StatefulSet`，有的没有开启，那么同一个 TidbCluster Custom Resource (CR) 不能在这些 TiDB Operator 之间切换。
- v1.1.10 开始支持此项功能

4.6.1 相关参数

为了支持部署多套 TiDB Operator，`tidb-operator` chart 中 `values.yaml` 文件里面添加了以下参数。

- `appendReleaseSuffix`

  如果配置为 `true`，部署时会自动为 `tidb-controller-manager` 和 `tidb-scheduler` ↪ 相关的资源名称添加后缀 `-{{ .Release.Name }}`，例如，通过 `helm install` ↪ `canary pingcap/tidb-operator ...` 命令部署的 `tidb-controller-manager` ↪ deployment 名称为：`tidb-controller-manager-canary`，如果要部署多套 TiDB Operator 需要开启此参数。

  默认值：`false`。

- controllerManager.create

  控制是否创建 tidb-controller-manager。

  默认值：true。

- controllerManager.selector

  配置 tidb-controller-manager 的 -selector 参数，用于根据 CR 的 label 筛选 tidb-controller-manager 控制的 CR，多个 selector 之间为 and 关系。

  默认值：[]，控制所有 CR。

  示例：

```
selector:
- canary-release=v1
- k1==v1
- k2!=v2
```

- scheduler.create

  控制是否创建 tidb-scheduler。

  默认值：true。

4.6.2 部署多套 TiDB Operator 分别控制不同 TiDB 集群

1. 部署第一套 TiDB Operator。

   参考部署 TiDB Operator 文档，在 values.yaml 中添加如下配置，部署第一套 TiDB Operator：

```
controllerManager:
  selector:
  - user=dev
```

2. 部署 TiDB 集群。

   1. 参考在 Kubernetes 中配置 TiDB 集群配置 TidbCluster CR，并配置 labels 匹配上一步中为 tidb-controller-manager 配置的 selector，例如：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic1
  labels:
    user: dev
spec:
  ...
```

   如果创建 TiDB 集群时没有设置 label，也可以通过如下命令设置：

```
kubectl -n ${namespace} label tidbcluster ${cluster_name} user=dev
```

2. 参考在 Kubernetes 中部署 TiDB 集群部署 TiDB 集群，并确认集群各组件正常启动。

3. 部署第二套 TiDB Operator。

   参考部署 TiDB Operator 文档，在 `values.yaml` 中添加如下配置，在不同的 **namespace** 中（例如 `tidb-admin-qa`）使用不同的 **Helm Release Name**（例如 `helm install tidb-operator-qa ...`）部署第二套 TiDB Operator (没有部署 `tidb-scheduler`)：

```
controllerManager:
  selector:
  - user=qa
appendReleaseSuffix: true
scheduler:
  create: false
advancedStatefulset:
  create: false
admissionWebhook:
  create: false
```

注意：

- 建议在单独的 namespace 部署新的 TiDB Operator。
- `appendReleaseSuffix` 需要设置为 `true`。
- 如果配置 `scheduler.create: true`，会创建一个名字为 `{{ .scheduler.schedulerName }}-{{.Release.Name}}` 的 scheduler，要使用这个 scheduler，需要配置 TidbCluster CR 中的 `spec.schedulerName` 为这个 scheduler。
- 由于不支持部署多套 AdvancedStatefulSet controller 和 AdmissionWebhook，需要配置 `advancedStatefulset.create: false` 和 `admissionWebhook.create: false`。

4. 部署 TiDB 集群。

   1. 参考在 Kubernetes 中配置 TiDB 集群配置 TidbCluster CR，并配置 `labels` 匹配上一步中为 `tidb-controller-manager` 配置的 `selector`，例如：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic2
```

```
  labels:
    user: qa
spec:
  ...
```

如果创建 TiDB 集群时没有设置 label，也可以通过如下命令设置：

```
kubectl -n ${namespace} label tidbcluster ${cluster_name} user=qa
```

2. 参考在 Kubernetes 中部署 TiDB 集群部署 TiDB 集群，并确认集群各组件正常启动。

5. 查看两套 TiDB Operator 的日志，确认两套 TiDB Operator 分别管理各自匹配 selector 的 TiDB 集群。

   示例：

   查看第一套 TiDB Operator `tidb-controller-manager` 的日志:

```
kubectl -n tidb-admin logs tidb-controller-manager-55b887bdc9-lzdwv
```

   Output

   查看第二套 TiDB Operator `tidb-controller-manager` 的日志:

```
kubectl -n tidb-admin-qa logs tidb-controller-manager-qa-5dfcd7f9-vll4c
```

   Output

   通过对比两套 TiDB Operator tidb-controller-manager 日志，第一套 TiDB Operator 仅管理 `tidb-cluster-1/basic1` 集群，第二套 TiDB Operator 仅管理 `tidb-cluster-2/basic2` 集群。

4.7 部署 TiDB 集群监控

4.7.1 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。

4.7.1.1 TiDB 集群的监控

TiDB 通过 Prometheus 和 Grafana 监控 TiDB 集群。在通过 TiDB Operator 创建新的 TiDB 集群时，可以对于每个 TiDB 集群，创建、配置一套独立的监控系统，与 TiDB 集群运行在同一 Namespace，包括 Prometheus 和 Grafana 两个组件。

在 TiDB 集群监控中有一些监控系统配置的细节可供参考。

在 v1.1 及更高版本的 TiDB Operator 中，可以通过简单的 CR 文件（即 TidbMonitor，可参考 tidb-operator 中的示例）来快速建立对 Kubernetes 集群上的 TiDB 集群的监控。

注意：

- 一个 TidbMonitor 只支持监控一个 TidbCluster。
- spec.clusters[0].name 需要配置为 TiDB 集群 TidbCluster 的名字。

4.7.1.1.1 持久化监控数据

可以在 TidbMonitor 中设置 spec.persistent 为 true 来持久化监控数据。开启此选项时应将 spec.storageClassName 设置为一个当前集群中已有的存储，并且此存储应当支持将数据持久化，否则会存在数据丢失的风险。配置示例如下：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: basic
spec:
  clusters:
    - name: basic
  persistent: true
  storageClassName: ${storageClassName}
  storage: 5G
  prometheus:
    baseImage: prom/prometheus
    version: v2.18.1
    service:
      type: NodePort
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: NodePort
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

你可以通过以下命令来确认 PVC 情况:

```
kubectl get pvc -l app.kubernetes.io/instance=basic,app.kubernetes.io/
    ↪ component=monitor -n ${namespace}
```

```
NAME           STATUS  VOLUME                                     CAPACITY  ACCESS
    ↪  MODES STORAGECLASS AGE
basic-monitor Bound   pvc-6db79253-cc9e-4730-bbba-ba987c29db6f 5G     RWO
    ↪           standard     51s
```

4.7.1.1.2 自定义 Prometheus 配置

用户可以通过自定义配置文件或增加额外的命令行参数，来自定义 Prometheus 配置。

使用自定义配置文件

1. 为用户自定义配置创建 ConfigMap 并将 `data` 部分的键名设置为 `prometheus-config` ↪ 。
2. 设置 `spec.prometheus.config.configMapRef.name` 与 `spec.prometheus.config.` ↪ `configMapRef.namespace` 为自定义 ConfigMap 的名称与所属的 namespace。

如需了解完整的配置示例，可参考 tidb-operator 中的示例。

增加额外的命令行参数

设置 `spec.prometheus.config.commandOptions` 为用于启动 Prometheus 的额外的命令行参数。

如需了解完整的配置示例，可参考 tidb-operator 中的示例。

> **注意：**
>
> 以下参数已由 TidbMonitor controller 自动设置，不支持通过 `commandOptions` 重复指定：
>
> - `config.file`
> - `log.level`
> - `web.enable-admin-api`
> - `web.enable-lifecycle`
> - `storage.tsdb.path`
> - `storage.tsdb.retention`
> - `storage.tsdb.max-block-duration`
> - `storage.tsdb.min-block-duration`

4.7.1.1.3 访问 Grafana 监控面板

可以通过 kubectl port-forward 访问 Grafana 监控面板：

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

然后在浏览器中打开 http://localhost:3000，默认用户名和密码都为 admin。

也可以设置 spec.grafana.service.type 为 NodePort 或者 LoadBalancer，通过 NodePort 或者 LoadBalancer 查看监控面板。

如果不需要使用 Grafana，可以在部署时将 TidbMonitor 中的 spec.grafana 部分删除。这一情况下需要使用其他已有或新部署的数据可视化工具直接访问监控数据来完成可视化。

4.7.1.1.4 访问 Prometheus 监控数据

对于需要直接访问监控数据的情况，可以通过 kubectl port-forward 来访问 Prometheus：

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-prometheus
    ↪ 9090:9090 &>/tmp/portforward-prometheus.log &
```

然后在浏览器中打开 http://localhost:9090，或通过客户端工具访问此地址即可。

也可以设置 spec.prometheus.service.type 为 NodePort 或者 LoadBalancer，通过 NodePort 或者 LoadBalancer 访问监控数据。

4.7.1.1.5 设置 kube-prometheus 与 AlertManager

在部分情况下，你可能需要 TidbMonitor 同时获取 Kubernetes 上的监控指标。你可以通过设置 TidbMonitor.Spec.kubePrometheusURL 来使其获取 kube-prometheus metrics。

同样的，你可以通过设置 TidbMonitor 来将监控推送警报至指定的 AlertManager。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: basic
spec:
  clusters:
    - name: basic
  kubePrometheusURL: http://prometheus-k8s.monitoring:9090
  alertmanagerURL: alertmanager-main.monitoring:9093
  prometheus:
    baseImage: prom/prometheus
    version: v2.18.1
    service:
      type: NodePort
```

```
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: NodePort
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

4.7.1.2 开启 Ingress

本节介绍如何为 TidbMonitor 开启 Ingress。Ingress 是一个 API 对象，负责管理集群中服务的外部访问。

4.7.1.2.1 环境准备

使用 Ingress 前，需要在 Kubernetes 集群中安装 Ingress 控制器，否则仅创建 Ingress 资源无效。你可能需要部署 Ingress 控制器，例如 ingress-nginx。你可以从许多 Ingress 控制器中进行选择。

更多关于 Ingress 环境准备，可以参考 Ingress 环境准备

4.7.1.2.2 使用 Ingress 访问 TidbMonitor

目前, TidbMonitor 提供了通过 Ingress 将 Prometheus/Grafana 服务暴露出去的方式，你可以通过 Ingress 文档了解更多关于 Ingress 的详情。

以下是一个开启了 Prometheus 与 Grafana Ingress 的 TidbMonitor 例子：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ingress-demo
spec:
  clusters:
    - name: demo
  persistent: false
  prometheus:
    baseImage: prom/prometheus
    version: v2.18.1
    ingress:
      hosts:
      - example.com
```

```
    annotations:
      foo: "bar"
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: ClusterIP
    ingress:
      hosts:
        - example.com
      annotations:
        foo: "bar"
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

你可以通过 `spec.prometheus.ingress.annotations` 与 `spec.grafana.ingress.annotations` 来设置对应的 Ingress Annotations 的设置。如果你使用的是默认的 Nginx Ingress 方案，你可以在 Nginx Ingress Controller Annotation 了解更多关于 Annotations 的详情。

`TidbMonitor` 的 Ingress 设置同样支持设置 TLS，以下是一个为 Ingress 设置 TLS 的例子。你可以通过 Ingress TLS 来了解更多关于 Ingress TLS 的资料。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ingress-demo
spec:
  clusters:
    - name: demo
  persistent: false
  prometheus:
    baseImage: prom/prometheus
    version: v2.18.1
    ingress:
      hosts:
      - example.com
      tls:
      - hosts:
        - example.com
        secretName: testsecret-tls
```

```
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: ClusterIP
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

TLS Secret 必须包含名为 tls.crt 和 tls.key 的密钥，这些密钥包含用于 TLS 的证书和私钥，例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: ${namespace}
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

在公有云 Kubernetes 集群中，通常可以配置 Loadbalancer 通过域名访问 Ingress。如果无法配置 Loadbalancer 服务，比如使用了 NodePort 作为 Ingress 的服务类型，可通过与如下命令等价的方式访问服务：

```
curl -H "Host: example.com" ${node_ip}:${NodePort}
```

4.7.1.3 告警配置

在随 TiDB 集群部署 Prometheus 时，会自动导入一些默认的告警规则，可以通过浏览器访问 Prometheus 的 Alerts 页面查看当前系统中的所有告警规则和状态。

目前支持自定义配置告警规则，可以参考下面步骤修改告警规则：

1. 在为 TiDB 集群部署监控的过程中，设置 `spec.reloader.service.type` 为 `NodePort` 或者 `LoadBalancer`。
2. 通过 `NodePort` 或者 `LoadBalancer` 访问 reloader 服务，点击上方 `Files` 选择要修改的告警规则文件进行修改，修改完成后 `Save`。

默认的 Prometheus 和告警配置不能发送告警消息，如需发送告警消息，可以使用任意支持 Prometheus 告警的工具与其集成。推荐通过 AlertManager 管理与发送告警消息。

如果在你的现有基础设施中已经有可用的 AlertManager 服务，可以参考设置 kube-prometheus 与 AlertManager 设置 spec.alertmanagerURL 配置其地址供 Prometheus 使用；如果没有可用的 AlertManager 服务，或者希望部署一套独立的服务，可以参考官方的说明部署。

4.7.2 TiDB Dashboard 指南

> **警告：**
>
> TiDB Dashboard 位于 PD 的 /dashboard 路径中。其他路径可能无法访问控制。

TiDB Dashboard 是从 TiDB 4.0 开始引入的专门用来帮助观察与诊断整个 TiDB 集群的可视化面板，你可以在 TiDB Dashboard 了解详情。本篇文章将介绍如何在 Kubernetes 环境下访问 TiDB Dashboard。

> **注意：**
>
> TiDB Operator 会为每一个 TiDB 集群启动一个 Discovery 服务。Discovery 服务会为每个 PD Pod 返回相应的启动参数，来辅助 PD 集群启动。此外，Discovery 服务也会发送代理请求到 TiDB Dashboard。本文档我们将通过 Discovery 服务访问 TiDB Dashboard。

4.7.2.1 前置条件

你需要使用 v1.1.1 版本及以上的 TiDB Operator 以及 4.0.1 版本及以上的 TiDB 集群，才能在 Kubernetes 环境中流畅使用 Dashboard。你需要在 TidbCluster 对象文件中通过以下方式开启 Dashboard 快捷访问:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  pd:
    enableDashboardInternalProxy: true
```

4.7.2.2 通过端口转发访问 TiDB Dashboard

> **注意：**
>
> 以下教程仅为演示如何快速访问 TiDB Dashboard，请勿在生产环境中直接使用以下方法。

TiDB Dashboard 目前在 4.0.0 版本及以上中已经内嵌在了 PD 组件中，你可以通过以下的例子在 Kubernetes 环境下快速部署一个 4.0.4 版本的 TiDB 集群。运行 `kubectl apply -f` 命令，将以下 yaml 文件部署到 Kubernetes 集群中。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    enableDashboardInternalProxy: true
    baseImage: pingcap/pd
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    service:
      type: ClusterIP
    config: {}
```

当集群创建完毕时，你可以通过以下指令将 TiDB Dashboard 暴露在本地机器：

```
kubectl port-forward svc/basic-discovery -n ${namespace} 10262:10262
```

然后在浏览器中访问 http://localhost:10262/dashboard 即可访问到 TiDB Dashboard。

> **注意：**
>
> port-forward 默认绑定 IP 地址 127.0.0.1。如果你需要使用其它 IP 地址访问运行 port-forward 命令的机器，可以通过 --address 选项指定需要绑定的 IP 地址。

4.7.2.3 通过 Ingress 访问 TiDB Dashboard

> **注意：**
>
> 我们推荐在生产环境、关键环境内使用 Ingress 来暴露 TiDB Dashboard 服务。

4.7.2.3.1 环境准备

使用 Ingress 前需要 Kubernetes 集群安装有 Ingress 控制器，仅创建 Ingress 资源无效。您可能需要部署 Ingress 控制器，例如 ingress-nginx。您可以从许多 Ingress 控制器 中进行选择。

4.7.2.3.2 使用 Ingress

你可以通过 Ingress 来将 TiDB Dashboard 服务暴露到 Kubernetes 集群外，从而在 Kubernetes 集群外通过 http/https 的方式访问服务。你可以通过 Ingress 了解更多关于 Ingress 的信息。以下是一个使用 Ingress 访问 TiDB Dashboard 的 yaml 文件例子。运行 kubectl apply -f 命令，将以下 yaml 文件部署到 Kubernetes 集群中。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: access-dashboard
  namespace: ${namespace}
spec:
  rules:
    - host: ${host}
      http:
        paths:
          - backend:
              serviceName: ${cluster_name}-discovery
```

```
            servicePort: 10262
          path: /dashboard
```

当部署了 Ingress 后，你可以在 Kubernetes 集群外通过 http://$%7Bhost%7D/dashboard 访问 TiDB Dashboard。

4.7.2.4 开启 Ingress TLS

Ingress 提供了 TLS 支持，你可以通过 Ingress TLS 了解更多。以下是一个使用 Ingress TLS 的例子，其中 `testsecret-tls` 包含了 `exmaple.com` 所需要的 `tls.crt` 与 `tls.key`：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: access-dashboard
  namespace: ${namespace}
spec:
  tls:
  - hosts:
    - ${host}
    secretName: testsecret-tls
  rules:
    - host: ${host}
      http:
        paths:
          - backend:
              serviceName: ${cluster_name}-discovery
              servicePort: 10262
            path: /dashboard
```

以下是 `testsecret-tls` 的一个例子:

```
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

当 Ingress 部署完成以后，你就可以通过 https://%7Bhost%7D/dashboard 访问 TiDB Dashboard。

4.7.2.4.1 使用 NodePort Service

由于 Ingress 必需使用域名访问，在某些场景下可能难以使用，此时可以通过添加一个 NodePort 类型的 Service 来访问和使用 TiDB Dashboard.

以下是一个使用 NodePort 类型的 Service 访问 TiDB Dashboard 的 yaml 文件例子。运行 kubectl apply -f 命令，将以下 yaml 文件部署到 Kubernetes 集群中。

```
apiVersion: v1
kind: Service
metadata:
  name: access-dashboard
  namespace: ${namespace}
spec:
  ports:
  - name: dashboard
    port: 10262
    protocol: TCP
    targetPort: 10262
  type: NodePort
  selector:
    app.kubernetes.io/component: discovery
    app.kubernetes.io/instance: ${cluster_name}
    app.kubernetes.io/name: tidb-cluster
```

当 Service 部署完成后，可以通过 https://%7BnodeIP%7D:%7BnodePort%7D/dashboard 访问 TiDB Dashboard, 其中 nodePort 默认由 Kubernetes 随机分配，也可以在 yaml 文件中指定一个可用的端口。

需要注意如果 PD Pod 数量超过 1 ，需要在 TidbCluster CR 中设置 spec.pd.
↪ enableDashboardInternalProxy: true 以保证正常访问 TiDB Dashboard。

4.7.2.5 更新 TiDB 集群

如果你是在一个已经运行的 TiDB 集群上进行更新来开启快捷访问 Dashboard 功能，以下两项配置都需要更新:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  configUpdateStrategy: RollingUpdate
  pd:
    enableDashboardInternalProxy: true
```

4.7.2.6 TiDB Operator 中不支持的 Dashboard 功能

TiDB Dashboard 中的部分功能会因为 kubernetes 的特殊环境而无法使用，包括以下功能：

1. 概况 -> 监控和告警 -> 查看监控项的链接无法正确跳转到 Grafana 监控页。如果需要访问 Grafana 监控，可以参考访问 Grafana 监控面板。
2. 日志搜索功能无法使用。如果需要查看对应组件的日志，可以使用 `kubectl logs ${pod_name} -n {namespace}` 查看对应组件的日志或者通过 Kubernetes 集群的日志服务查看。
3. 集群信息 -> 主机的磁盘容量，磁盘使用率无法正确显示。可以通过TidbMonitor 监控面板 的各组件 dashboards 查看各个组件的磁盘使用或者部署Kubernetes 宿主机 Grafana 监控 查看 Kubernetes 节点的磁盘使用情况。

5 安全

5.1 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。TiDB Operator 从 v1.1 开始已经支持为 Kubernetes 上 TiDB 集群开启 MySQL 客户端 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：`${cluster_name}-tidb-server-secret` 和 `${cluster_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。

2. 部署集群，设置 `.spec.tidb.tlsClient.enabled` 属性为 `true`；
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；
- 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。

5.1.1 第一步：为 TiDB 集群颁发两套证书

5.1.1.1 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",
      "${cluster_name}-tidb",
      "${cluster_name}-tidb.${namespace}",
      "${cluster_name}-tidb.${namespace}.svc",
      "*.${cluster_name}-tidb",
      "*.${cluster_name}-tidb.${namespace}",
      "*.${cluster_name}-tidb.${namespace}.svc",
      "*.${cluster_name}-tidb-peer",
      "*.${cluster_name}-tidb-peer.${namespace}",
```

```
      "*.${cluster_name}-tidb-peer.${namespace}.svc"
    ],
...
```

其中 `${cluster_name}` 为集群的名字，`${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 ${cluster_name}-tidb-server-secret --
       ↪ namespace=${namespace} --from-file=tls.crt=server.pem --from-file
       ↪ =tls.key=server-key.pem --from-file=ca.crt=ca.pem
   kubectl create secret generic ${cluster_name}-tidb-client-secret --
       ↪ namespace=${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 (目前有 TidbInitializer 会访问 TiDB Server 来设置密码或者一些初始化操作)。

5.1.1.2 使用 cert-manager 颁发证书

1. 安装 cert-manager。

   请参考官网安装：cert-manager installation in 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/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1alpha2
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/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-tidb-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/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-server-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-server-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "TiDB Server"
  usages:
    - server auth
  dnsNames:
    - "${cluster_name}-tidb"
    - "${cluster_name}-tidb.${namespace}"
    - "${cluster_name}-tidb.${namespace}.svc"
    - "*.${cluster_name}-tidb"
    - "*.${cluster_name}-tidb.${namespace}"
    - "*.${cluster_name}-tidb.${namespace}.svc"
    - "*.${cluster_name}-tidb-peer"
    - "*.${cluster_name}-tidb-peer.${namespace}"
    - "*.${cluster_name}-tidb-peer.${namespace}.svc"
  ipAddresses:
    - 127.0.0.1
    - ::1
  issuerRef:
```

```
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 ${cluster_name} 为集群的名字：

- spec.secretName 请设置为 ${cluster_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：
- ${cluster_name}-tidb
- ${cluster_name}-tidb.${namespace}
- ${cluster_name}-tidb.${namespace}.svc
- *.${cluster_name}-tidb
- *.${cluster_name}-tidb.${namespace}
- *.${cluster_name}-tidb.${namespace}.svc
- *.${cluster_name}-tidb-peer
- *.${cluster_name}-tidb-peer.${namespace}
- *.${cluster_name}-tidb-peer.${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 ${cluster_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/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "TiDB Client"
```

```
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 ${cluster_name} 为集群的名字：

- spec.secretName 请设置为 ${cluster_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 ${cluster_name}-tidb-client
↪ -secret 的 Secret 对象供 TiDB Client 使用。

5. 创建多套 Client 端证书（可选）。

   TiDB Operator 集群内部有 4 个组件需要请求 TiDB Server，当开启 TLS 验证后，这些组件可以使用证书来请求 TiDB Server，每个组件都可以使用单独的证书。这些组件有：

   - TidbInitializer
   - PD Dashboard
   - Backup
   - Restore

   如需要使用 TiDB Lightning 恢复 Kubernetes 上的集群数据，则也可以为其中的 TiDB Lightning 组件生成 Client 端证书。

   下面就来生成这些组件的 Client 证书。

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

      ```
      apiVersion: cert-manager.io/v1alpha2
      kind: Certificate
      metadata:
        name: ${cluster_name}-tidb-initializer-client-secret
        namespace: ${namespace}
      spec:
        secretName: ${cluster_name}-tidb-initializer-client-secret
        duration: 8760h # 365d
      ```

```
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "TiDB Initializer client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-pd-dashboard-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-pd-dashboard-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "PD Dashboard client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-backup-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-backup-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "Backup client"
  usages:
    - client auth
  issuerRef:
```

```
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-restore-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-restore-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "Restore client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 `${cluster_name}` 为集群的名字：

- `spec.secretName` 请设置为 `${cluster_name}-${component}-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。

如需要为 TiDB Lignting 组件生成 Client 端证书，则可以使用以下内容并通过在 TiDB Lightning 的 Helm Chart `values.yaml` 中设置 `tlsCluster.tlsClientSecretName` 为 `${cluster_name}-lightning-client-secret`：

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-lightning-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-lightning-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
```

```
  commonName: "Lightning client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

2. 通过执行下面的命令来创建证书：

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

3. 创建这些对象以后，cert-manager 会生成 4 个 Secret 对象供上面四个组件使用。

> **注意：**
> TiDB Server 的 TLS 兼容 MySQL 协议。当证书内容发生改变后，需要管理员手动执行 SQL 语句 `alter instance reload tls` 进行刷新。

5.1.2 第二步：部署 TiDB 集群

接下来将会创建一个 TiDB 集群，并且执行以下步骤：

- 开启 MySQL 客户端 TLS；
- 对集群进行初始化（这里创建了一个数据库 app）；
- 创建一个 Backup 对象对集群进行备份；
- 创建一个 Restore 对象对进群进行恢复；
- TidbInitializer，PD Dashboard，Backup 以及 Restore 分别使用单独的 Client 证书（用 `tlsClientSecretName` 指定）。

1. 创建三个 `.yaml` 文件：

   - `tidb-cluster.yaml`:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
 name: ${cluster_name}
 namespace: ${namespace}
spec:
 version: v5.0.6
 timezone: UTC
 pvReclaimPolicy: Retain
 pd:
```

```
  baseImage: pingcap/pd
  replicas: 1
  requests:
    storage: "1Gi"
  config: {}
  tlsClientSecretName: ${cluster_name}-pd-dashboard-client-secret
 tikv:
  baseImage: pingcap/tikv
  replicas: 1
  requests:
    storage: "1Gi"
  config: {}
 tidb:
  baseImage: pingcap/tidb
  replicas: 1
  service:
    type: ClusterIP
  config: {}
  tlsClient:
    enabled: true
---
apiVersion: pingcap.com/v1alpha1
kind: TidbInitializer
metadata:
 name: ${cluster_name}-init
 namespace: ${namespace}
spec:
 image: tnir/mysqlclient
 cluster:
  namespace: ${namespace}
  name: ${cluster_name}
 initSql: |-
  create database app;
 tlsClientSecretName: ${cluster_name}-tidb-initializer-client-
    ↪ secret
```

- backup.yaml:

```
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: ${cluster_name}-backup
  namespace: ${namespace}
spec:
  backupType: full
```

```
  br:
    cluster: ${cluster_name}
    clusterNamespace: ${namespace}
    sendCredToTikv: true
  from:
    host: ${host}
    secretName: ${tidb_secret}
    port: 4000
    user: root
    tlsClientSecretName: ${cluster_name}-backup-client-secret
  s3:
    provider: aws
    region: ${my_region}
    secretName: ${s3_secret}
    bucket: ${my_bucket}
    prefix: ${my_folder}
```

- restore.yaml:

```
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: ${cluster_name}-restore
  namespace: ${namespace}
spec:
  backupType: full
  br:
    cluster: ${cluster_name}
    clusterNamespace: ${namespace}
    sendCredToTikv: true
  to:
    host: ${host}
    secretName: ${tidb_secret}
    port: 4000
    user: root
    tlsClientSecretName: ${cluster_name}-restore-client-secret
  s3:
    provider: aws
    region: ${my_region}
    secretName: ${s3_secret}
    bucket: ${my_bucket}
    prefix: ${my_folder}
```

其中 `${cluster_name}` 为集群的名字，`${namespace}` 为 TiDB 集群部署的命名空间。通过设置 `spec.tidb.tlsClient.enabled` 属性为 `true` 来开启 MySQL 客户端

TLS。

2. 部署 TiDB 集群：

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

3. 集群备份：

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

4. 集群恢复：

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

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

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

> 注意：
>
> MySQL 8.0 默认认证插件从 `mysql_native_password` 更新为 `caching_sha2_password`，因此如果使用 MySQL 8.0 客户端访问 TiDB 服务（TiDB 版本 $<$ v4.0.7），并且用户账户有配置密码，需要显示指定 `--default-auth=mysql_native_password` 参数。

最后请参考 官网文档 来验证是否正确开启了 TLS。

5.2 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。TiDB Operator 从 v1.1 开始已经支持为 Kubernetes 上 TiDB 集群组件间开启 TLS。开启步骤为：

1. 为即将被创建的 TiDB 集群的每个组件生成证书：
    - 为 PD/TiKV/TiDB/Pump/Drainer/TiFlash/TiKV Importer/TiDB Lightning 组件分别创建一套 Server 端证书，保存为 Kubernetes Secret 对象：`${cluster_name}-${component_name}-cluster-secret`
    - 为它们的各种客户端创建一套共用的 Client 端证书，保存为 Kubernetes Secret 对象：`${cluster_name}-cluster-client-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，设置 `.spec.tlsCluster.enabled` 属性为 `true`；
3. 配置 `pd-ctl`，`tikv-ctl` 连接集群。

> 注意：
>
> - TiDB v4.0.5, TiDB Operator v1.1.4 及以上版本支持 TiFlash 开启 TLS。
> - TiDB v4.0.3, TiDB Operator v1.1.3 及以上版本支持 TiCDC 开启 TLS。

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；
- 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。

5.2.1 第一步：为 TiDB 集群各个组件生成证书

5.2.1.1 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：

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

   这里需要为每个 TiDB 集群的组件生成一套 Server 端证书。

   - PD Server 端证书

     首先生成默认的 `pd-server.json` 文件：

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

     然后编辑这个文件，修改 `CN`，`hosts` 属性：

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

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

其中 ${cluster_name} 为集群的名字，${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 PD Server 端证书：

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

- TiKV Server 端证书

  首先生成默认的 tikv-server.json 文件：

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

  然后编辑这个文件，修改 CN，hosts 属性：

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

  其中 ${cluster_name} 为集群的名字，${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 TiKV Server 端证书：

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

- TiDB Server 端证书

  首先生成默认的 tidb-server.json 文件：

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

然后编辑这个文件，修改 CN，hosts 属性：

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

其中 ${cluster_name} 为集群的名字，${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 TiDB Server 端证书：

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

- Pump Server 端证书

  首先生成默认的 pump-server.json 文件：

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

  然后编辑这个文件，修改 CN，hosts 属性：

```
...
   "CN": "TiDB",
   "hosts": [
     "127.0.0.1",
     "::1",
     "*.${cluster_name}-pump",
     "*.${cluster_name}-pump.${namespace}",
     "*.${cluster_name}-pump.${namespace}.svc"
   ],
...
```

  其中 ${cluster_name} 为集群的名字，${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 Pump Server 端证书：

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

- Drainer Server 端证书

  首先生成默认的 drainer-server.json 文件：

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

  然后编辑这个文件，修改 CN，hosts 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "<hosts 列表请参考下面描述>"
    ],
...
```

  现在 Drainer 组件是通过 Helm 来部署的，根据 values.yaml 文件配置方式不同，所需要填写的 hosts 字段也不相同。

  如果部署的时候设置 drainerName 属性，像下面这样：

```
...
# Change the name of the statefulset and pod
# The default is clusterName-ReleaseName-drainer
# Do not change the name of an existing running drainer: this is
    ↪ unsupported.
drainerName: my-drainer
...
```

  那么就这样配置 hosts 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "*.${drainer_name}",
      "*.${drainer_name}.${namespace}",
      "*.${drainer_name}.${namespace}.svc"
    ],
...
```

  如果部署的时候没有设置 drainerName 属性，需要这样配置 hosts 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "*.${cluster_name}-${release_name}-drainer",
      "*.${cluster_name}-${release_name}-drainer.${namespace}",
      "*.${cluster_name}-${release_name}-drainer.${namespace}.svc"
    ],
...
```

其中 `${cluster_name}` 为集群的名字，`${namespace}` 为 TiDB 集群部署的命名空间，`${release_name}` 是 `helm install` 时候填写的 `release name`，`${drainer_name}` 为 `values.yaml` 文件里的 `drainerName`，用户也可以添加自定义 `hosts`。

最后生成 Drainer Server 端证书：

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

- TiCDC Server 端证书

  首先生成默认的 `ticdc-server.json` 文件：

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

  然后编辑这个文件，修改 `CN`，`hosts` 属性：

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

  其中 `${cluster_name}` 为集群的名字，`${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 TiCDC Server 端证书：

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

- TiFlash Server 端证书

  首先生成默认的 tiflash-server.json 文件：

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

  然后编辑这个文件，修改 CN、hosts 属性：

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

  其中 ${cluster_name} 为集群的名字，${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 TiFlash Server 端证书：

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

- TiKV Importer Server 端证书

  如需要使用 TiDB Lightning 恢复 Kubernetes 上的集群数据，则需要为其中的 TiKV Importer 组件生成如下的 Server 端证书。

  首先生成默认的 importer-server.json 文件：

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

  然后编辑这个文件，修改 CN、hosts 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${cluster_name}-importer",
      "${cluster_name}-importer.${namespace}",
      "${cluster_name}-importer.${namespace}.svc",
      "*.${cluster_name}-importer",
      "*.${cluster_name}-importer.${namespace}",
      "*.${cluster_name}-importer.${namespace}.svc"
    ],
...
```

其中 `${cluster_name}` 为集群的名字，`${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 TiKV Importer Server 端证书：

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

- TiDB Lightning Server 端证书

  如需要使用 TiDB Lightning 恢复 Kubernetes 上的集群数据，则需要为其中的 TiDB Lightning 组件生成如下的 Server 端证书。

  首先生成默认的 `lightning-server.json` 文件：

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

  然后编辑这个文件，修改 `CN`、`hosts` 属性：

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${cluster_name}-lightning",
      "${cluster_name}-lightning.${namespace}",
      "${cluster_name}-lightning.${namespace}.svc"
    ],
...
```

  其中 `${cluster_name}` 为集群的名字，`${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 TiDB Lightning Server 端证书：

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
    ↪ -profile=internal lightning-server.json | cfssljson -bare
    ↪ lightning-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",
    "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。

   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 ${cluster_name}-pd-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=pd-server.pem --from-
    ↪ file=tls.key=pd-server-key.pem --from-file=ca.crt=ca.pem
```

   TiKV 集群证书 Secret：

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

   TiDB 集群证书 Secret：

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

   Pump 集群证书 Secret：

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

Drainer 集群证书 Secret：

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

TiCDC 集群证书 Secret：

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

TiFlash 集群证书 Secret：

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

TiKV Importer 集群证书 Secret：

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

TiDB Lightning 集群证书 Secret：

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

Client 证书 Secret：

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

这里给 PD/TiKV/TiDB/Pump/Drainer 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。

5.2.1.2 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 in 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/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1alpha2
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/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-tidb-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 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。

   根据官网文档：Enable TLS Authentication，我们需要为每个组件创建一个 Server 端证书，并且为他们的 Client 创建一套公用的 Client 端证书。

   - PD 组件的 Server 端证书。

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-pd-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-pd-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-pd"
  - "${cluster_name}-pd.${namespace}"
  - "${cluster_name}-pd.${namespace}.svc"
  - "${cluster_name}-pd-peer"
  - "${cluster_name}-pd-peer.${namespace}"
  - "${cluster_name}-pd-peer.${namespace}.svc"
  - "*.${cluster_name}-pd-peer"
  - "*.${cluster_name}-pd-peer.${namespace}"
  - "*.${cluster_name}-pd-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
```

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

其中 ${cluster_name} 为集群的名字：

- spec.secretName 请设置为 ${cluster_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：
- ${cluster_name}-pd
- ${cluster_name}-pd.${namespace}
- ${cluster_name}-pd.${namespace}.svc
- ${cluster_name}-pd-pee
- ${cluster_name}-pd-peer.${namespace}
- ${cluster_name}-pd-peer.${namespace}.svc
- *.${cluster_name}-pd-peer
- *.${cluster_name}-pd-peer.${namespace}
- *.${cluster_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 ${cluster_name}-pd-cluster-secret 的 Secret 对象供 TiDB 集群的 PD 组件使用。

- TiKV 组件的 Server 端证书。

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tikv-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tikv-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-tikv"
  - "${cluster_name}-tikv.${namespace}"
  - "${cluster_name}-tikv.${namespace}.svc"
  - "${cluster_name}-tikv-peer"
```

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

其中 ${cluster_name} 为集群的名字：

- spec.secretName 请设置为 ${cluster_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：
- ${cluster_name}-tikv
- ${cluster_name}-tikv.${namespace}
- ${cluster_name}-tikv.${namespace}.svc
- ${cluster_name}-tikv-peer
- ${cluster_name}-tikv-peer.${namespace}
- ${cluster_name}-tikv-peer.${namespace}.svc
- *.${cluster_name}-tikv-peer
- *.${cluster_name}-tikv-peer.${namespace}
- *.${cluster_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 ${cluster_name}-tikv-cluster-secret 的 Secret 对象供 TiDB 集群的 TiKV 组件使用。

- TiDB 组件的 Server 端证书。

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
```

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

其中 `${cluster_name}` 为集群的名字：

- `spec.secretName` 请设置为 `${cluster_name}-tidb-cluster-secret`；
- `usages` 请添加上 `server auth` 和 `client auth`；
- `dnsNames` 需要填写这些 DNS，根据需要可以填写其他 DNS：
- `${cluster_name}-tidb`
- `${cluster_name}-tidb.${namespace}`
- `${cluster_name}-tidb.${namespace}.svc`
- `${cluster_name}-tidb-peer`
- `${cluster_name}-tidb-peer.${namespace}`
- `${cluster_name}-tidb-peer.${namespace}.svc`
- `*.${cluster_name}-tidb-peer`
- `*.${cluster_name}-tidb-peer.${namespace}`
- `*.${cluster_name}-tidb-peer.${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 `${cluster_name}-tidb-cluster-secret` 的 Secret 对象供 TiDB 集群的 TiDB 组件使用。

- Pump 组件的 Server 端证书。

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-pump-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-pump-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${cluster_name}-pump"
  - "*.${cluster_name}-pump.${namespace}"
  - "*.${cluster_name}-pump.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 `${cluster_name}` 为集群的名字：

- `spec.secretName` 请设置为 `${cluster_name}-pump-cluster-secret`；
- `usages` 请添加上 `server auth` 和 `client auth`；
- `dnsNames` 需要填写这些 DNS，根据需要可以填写其他 DNS：
- `*.${cluster_name}-pump`
- `*.${cluster_name}-pump.${namespace}`
- `*.${cluster_name}-pump.${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 `${cluster_name}-pump-cluster-secret` 的 Secret 对象供 TiDB 集群的 Pump 组件使用。

- Drainer 组件的 Server 端证书。
  现在 Drainer 组件是通过 Helm 来部署的，根据 `values.yaml` 文件配置方式不

同，所需要填写的 dnsNames 字段也不相同。

如果部署的时候设置 drainerName 属性，像下面这样：

```
...
# Change the name of the statefulset and pod
# The default is clusterName-ReleaseName-drainer
# Do not change the name of an existing running drainer: this is
    ↪ unsupported.
drainerName: my-drainer
...
```

那么就需要这样配置证书：

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-drainer-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-drainer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${drainer_name}"
  - "*.${drainer_name}.${namespace}"
  - "*.${drainer_name}.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

如果部署的时候没有设置 drainerName 属性，需要这样配置 dnsNames 属性：

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-drainer-cluster-secret
  namespace: ${namespace}
```

```
spec:
  secretName: ${cluster_name}-drainer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${cluster_name}-${release_name}-drainer"
  - "*.${cluster_name}-${release_name}-drainer.${namespace}"
  - "*.${cluster_name}-${release_name}-drainer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 ${cluster_name} 为集群的名字，${namespace} 为 TiDB 集群部署的命名空间，${release_name} 是 helm install 时候填写的 release name，${drainer_name} 为 values.yaml 文件里的 drainerName，用户也可以添加自定义 dnsNames。

- spec.secretName 请设置为 ${cluster_name}-drainer-cluster-secret；
- usages 请添加上 server auth 和 client auth；
- dnsNames 请参考上面的描述；
- 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 ${cluster_name}-drainer-cluster-secret 的 Secret 对象供 TiDB 集群的 Drainer 组件使用。

- TiCDC 组件的 Server 端证书。

TiCDC 从 v4.0.3 版本开始支持 TLS，TiDB Operator v1.1.3 版本同步支持 TiCDC 开启 TLS 功能。

```
``` yaml
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
```
```

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

其中 `${cluster_name}` 为集群的名字：

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

```
  - `*.${cluster_name}-ticdc-peer.${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](https://cert-manager.io/docs/
    ↪ reference/api-docs/#cert-manager.io/v1alpha2.CertificateSpec)。

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

- TiFlash 组件的 Server 端证书。

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tiflash-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tiflash-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-tiflash"
  - "${cluster_name}-tiflash.${namespace}"
  - "${cluster_name}-tiflash.${namespace}.svc"
  - "${cluster_name}-tiflash-peer"
  - "${cluster_name}-tiflash-peer.${namespace}"
  - "${cluster_name}-tiflash-peer.${namespace}.svc"
  - "*.${cluster_name}-tiflash-peer"
  - "*.${cluster_name}-tiflash-peer.${namespace}"
  - "*.${cluster_name}-tiflash-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 ${cluster_name} 为集群的名字：

- spec.secretName 请设置为 ${cluster_name}-tiflash-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：
  * ${cluster_name}-tiflash
  * ${cluster_name}-tiflash.${namespace}
  * ${cluster_name}-tiflash.${namespace}.svc
  * ${cluster_name}-tiflash-peer
  * ${cluster_name}-tiflash-peer.${namespace}
  * ${cluster_name}-tiflash-peer.${namespace}.svc
  * *.${cluster_name}-tiflash-peer
  * *.${cluster_name}-tiflash-peer.${namespace}
  * *.${cluster_name}-tiflash-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 ${cluster_name}-tiflash-cluster-secret 的 Secret 对象供 TiDB 集群的 TiFlash 组件使用。

- TiKV Importer 组件的 Server 端证书。

如需要使用 TiDB Lightning 恢复 Kubernetes 上的集群数据，则需要为其中的 TiKV Importer 组件生成如下的 Server 端证书。

```yaml
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-importer-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-importer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-importer"
```

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

其中 `${cluster_name}` 为集群的名字：

- `spec.secretName` 请设置为 `${cluster_name}-importer-cluster-secret`；
- `usages` 请添加上 `server auth` 和 `client auth`；
- `dnsNames` 需要填写这些 DNS，根据需要可以填写其他 DNS：
    - `${cluster_name}-importer`
    - `${cluster_name}-importer.${namespace}`
    - `${cluster_name}-importer.${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](https://cert-manager.io/docs/reference/api-docs/#cert-manager.io/v1alpha2.CertificateSpec)。

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

- TiDB Lightning 组件的 Server 端证书。

如需要使用 TiDB Lightning 恢复 Kubernetes 上的集群数据，则需要为其中的 TiDB Lightning 组件生成如下的 Server 端证书。

```yaml
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-lightning-cluster-secret
  namespace: ${namespace}
```

```
spec:
  secretName: ${cluster_name}-lightning-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-lightning"
  - "${cluster_name}-lightning.${namespace}"
  - "${cluster_name}-lightning.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 `${cluster_name}` 为集群的名字：

- `spec.secretName` 请设置为 `${cluster_name}-lightning-cluster-secret `;
- `usages` 请添加上 `server auth` 和 `client auth`;
- `dnsNames` 需要填写这些 DNS，根据需要可以填写其他 DNS：
    - `${cluster_name}-lightning`
    - `${cluster_name}-lightning.${namespace}`
    - `${cluster_name}-lightning.${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](https://cert-manager.io/docs/reference/api-docs/#cert-manager.io/v1alpha2.CertificateSpec)。

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

- 一套 TiDB 集群组件的 Client 端证书。

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-cluster-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-cluster-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

其中 `${cluster_name}` 为集群的名字：

- `spec.secretName` 请设置为 `${cluster_name}-cluster-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。

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

5.2.2 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：**
>
> 目前 PD 的 `cert-allowed-cn` 配置项只能设置一个值。因此所有 `Certificate` 对象的 `commonName` 都要设置成同样一个值。

在这一步中，需要完成以下操作：

- 创建一套 TiDB 集群

- 为 TiDB 组件间开启 TLS，并开启 CN 验证
- 部署一套监控系统
- 部署 Pump 组件，并开启 CN 验证

1. 创建一套 TiDB 集群 (监控系统和 Pump 组件已包含在内):

   创建 `tidb-cluster.yaml` 文件：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
 name: ${cluster_name}
 namespace: ${namespace}
spec:
 tlsCluster:
   enabled: true
 version: v5.0.6
 timezone: UTC
 pvReclaimPolicy: Retain
 pd:
   baseImage: pingcap/pd
   replicas: 1
   requests:
     storage: "1Gi"
   config:
     security:
       cert-allowed-cn:
         - TiDB
 tikv:
   baseImage: pingcap/tikv
   replicas: 1
   requests:
     storage: "1Gi"
   config:
     security:
       cert-allowed-cn:
         - TiDB
 tidb:
   baseImage: pingcap/tidb
   replicas: 1
   service:
     type: ClusterIP
   config:
     security:
       cluster-verify-cn:
         - TiDB
```

```
 pump:
   baseImage: pingcap/tidb-binlog
   replicas: 1
   requests:
     storage: "1Gi"
   config:
     security:
       cert-allowed-cn:
         - TiDB
---
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
 name: ${cluster_name}
 namespace: ${namespace}
spec:
 clusters:
 - name: ${cluster_name}
 prometheus:
   baseImage: prom/prometheus
   version: v2.11.1
 grafana:
   baseImage: grafana/grafana
   version: 6.0.1
 initializer:
   baseImage: pingcap/tidb-monitor-initializer
   version: v5.0.6
 reloader:
   baseImage: pingcap/tidb-monitor-reloader
   version: v1.0.1
 imagePullPolicy: IfNotPresent
```

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

2. 创建 Drainer 组件并开启 TLS 以及 CN 验证。

   - **第一种方式：创建 Drainer 的时候设置 `drainerName`：**
     编辑 values.yaml 文件，设置好 drainer-name，并将 TLS 功能打开：

```
...
drainerName: ${drainer_name}
tlsCluster:
  enabled: true
  certAllowedCN:
    - TiDB
...
```

然后部署 Drainer 集群：

```
helm install ${release_name} pingcap/tidb-drainer --namespace=${
    ↪ namespace} --version=${helm_version} -f values.yaml
```

- 第二种方式：创建 Drainer 的时候不设置 drainerName：
  编辑 values.yaml 文件，将 TLS 功能打开：

```
...
tlsCluster:
  enabled: true
  certAllowedCN:
    - TiDB
...
```

然后部署 Drainer 集群：

```
helm install ${release_name} pingcap/tidb-drainer --namespace=${
    ↪ namespace} --version=${helm_version} -f values.yaml
```

3. 创建 Backup/Restore 资源对象。

- 创建 backup.yaml 文件：

```
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: ${cluster_name}-backup
  namespace: ${namespace}
spec:
  backupType: full
  br:
    cluster: ${cluster_name}
    clusterNamespace: ${namespace}
    sendCredToTikv: true
  from:
    host: ${host}
    secretName: ${tidb_secret}
    port: 4000
    user: root
  s3:
    provider: aws
    region: ${my_region}
    secretName: ${s3_secret}
    bucket: ${my_bucket}
    prefix: ${my_folder}
```

然后部署 Backup：

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

- 创建 `restore.yaml` 文件：

```
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: ${cluster_name}-restore
  namespace: ${namespace}
spec:
  backupType: full
  br:
    cluster: ${cluster_name}
    clusterNamespace: ${namespace}
    sendCredToTikv: true
  to:
    host: ${host}
    secretName: ${tidb_secret}
    port: 4000
    user: root
  s3:
    provider: aws
    region: ${my_region}
    secretName: ${s3_secret}
    bucket: ${my_bucket}
    prefix: ${my_folder}
```

然后部署 Restore：

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

5.2.3 第三步：配置 pd-ctl、tikv-ctl 连接集群

1. 挂载证书。

   通过下面命令配置 `spec.pd.mountClusterClientSecret: true` 和 `spec.tikv.mountClusterClientSecret: true`：

```
kubectl edit tc ${cluster_name} -n ${namespace}
```

> 注意：
>
> - 上面配置改动会滚动升级 PD 和 TiKV 集群。
> - 上面配置从 TiDB Operator v1.1.5 开始支持。

2. 使用 pd-ctl 连接集群。

   进入 PD Pod：

```
kubectl exec -it ${cluster_name}-pd-0 -n ${namespace} sh
```

   使用 pd-ctl：

```
cd /var/lib/cluster-client-tls
/pd-ctl --cacert=ca.crt --cert=tls.crt --key=tls.key -u https
    ↪ ://127.0.0.1:2379 member
```

3. 使用 tikv-ctl 连接集群。

   进入 TiKV Pod：

```
kubectl exec -it ${cluster_name}-tikv-0 -n ${namespace} sh
```

   使用 tikv-ctl：

```
cd /var/lib/cluster-client-tls
/tikv-ctl --ca-path=ca.crt --cert-path=tls.crt --key-path=tls.key --
    ↪ host 127.0.0.1:20160 cluster
```

5.3 使用 TiCDC 组件同步数据到开启 TLS 的下游服务

本文介绍在 Kubernetes 上如何让 TiCDC 组件同步数据到开启 TLS 的下游服务。

5.3.1 准备条件

在开始之前，请进行以下准备工作：

- 部署一个下游服务，并开启客户端 TLS 认证。
- 生成客户端访问下游服务所需要的密钥文件。

5.3.2 TiCDC 同步数据到开启 TLS 的下游服务

1. 创建一个 Kubernetes Secret 对象，此对象需要包含用于访问下游服务的客户端 TLS 证书。证书来自于你为客户端生成的密钥文件。

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

2. 挂载证书文件到 TiCDC Pod。

- 如果你还未部署 TiDB 集群，在 TidbCluster CR 定义中添加 `spec.ticdc.tlsClientSecretNames` 字段，然后部署 TiDB 集群。
- 如果你已经部署了 TiDB 集群，执行 `kubectl edit tc ${cluster_name} -n ${cluster_namespace}`，并添加 `spec.ticdc.tlsClientSecretNames` 字段，然后等待 TiCDC 的 Pod 自动滚动更新。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${cluster_name}
  namespace: ${cluster_namespace}
spec:
  # ...
  ticdc:
    baseImage: pingcap/ticdc
    version: "v5.0.1"
    # ...
    tlsClientSecretNames:
    - ${secret_name}
```

TiCDC Pod 运行后，创建的 Kubernetes Secret 对象会被挂载到 TiCDC 的 Pod。你可以在 Pod 内的 `/var/lib/sink-tls/${secret_name}` 目录找到被挂载的密钥文件。

3. 通过 `cdc cli` 工具创建同步任务。

```
kubectl exec ${cluster_name}-ticdc-0 -- /cdc cli changefeed create --pd
    ↪ =https://${cluster_name}-pd:2379 --sink-uri="mysql://${user}:{
    ↪ $password}@${downstream_service}/?ssl-ca=/var/lib/sink-tls/${
    ↪ secret_name}/ca.crt&ssl-cert=/var/lib/sink-tls/${secret_name}/tls
    ↪ .crt&ssl-key=/var/lib/sink-tls/${secret_name}/tls.key"
```

5.4 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。

5.4.1 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。

5.4.1.1 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 ${cluster_name}-pd-cluster-secret --
    ↪ namespace=${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 ${cluster_name}-tikv-cluster-secret --
    ↪ namespace=${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 ${cluster_name}-tidb-cluster-secret --
    ↪ namespace=${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 -
```

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

其中 ${cluster_name} 为集群的名字，${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 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。

5.4.1.2 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
cfssl gencert -ca=ca.new.pem -ca-key=ca-key.new.pem -config=ca-config.
    ↪ json -profile=client client.json | cfssljson -bare client
```

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

其中 `${cluster_name}` 为集群的名字，`${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 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。

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

其中 `${cluster_name}` 为集群的名字，`${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。

5.4.2 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。

5.4.2.1 更新和替换 CA 证书及组件间证书

使用 cert-manager 颁发证书时，通过指定 `Certificate` 资源的 `spec.renewBefore` 可由 cert-manager 在证书过期之前自动进行更新。

但 cert-manager 虽然能自动更新 CA 证书及对应的 Kubernetes Secret 对象，但目前并不支持将新旧 CA 证书合并为组合 CA 证书以同时接受新旧 CA 证书签发的证书。因此，在更新和替换 CA 证书的过程中，会出现集群组件间 TLS 无法互相认证的问题。

> **警告：**
>
> 由于组件间无法同时接受新旧 CA 签发的证书，因此在更新和替换证书的过程中需要重建部分组件的 Pod，这可能会引起部分访问 TiDB 集群的请求失败。

相应的更新和替换 PD、TiKV、TiDB 的 CA 证书及组件间证书的步骤如下。

1. 由 cert-manager 在证书过期之前自动更新 CA 证书及 Kubernetes Secret 对象 `${cluster_name}-ca-secret`。

   其中 `${cluster_name}` 为集群的名字。

   若要手动更新 CA 证书，可直接删除相应的 Kubernetes Secret 对象后触发 cert-manager 重新生成 CA 证书。

2. 删除各组件对应证书的 Kubernetes Secret 对象。

```
kubectl delete secret ${cluster_name}-pd-cluster-secret --namespace=${namespace}
kubectl delete secret ${cluster_name}-tikv-cluster-secret --namespace=${namespace}
kubectl delete secret ${cluster_name}-tidb-cluster-secret --namespace=${namespace}
kubectl delete secret ${cluster_name}-cluster-client-secret --namespace=${namespace}
```

   其中 `${cluster_name}` 为集群的名字，`${namespace}` 为 TiDB 集群部署的命名空间。

3. 等待 cert-manager 基于新的 CA 证书为各组件颁发新的证书。

   观察 `kubectl get secret --namespace=${namespace}` 的输出，直到所有组件对应的 Kubernetes Secret 对象都被创建。

4. 依次强制重建 PD、TiKV 与 TiDB 组件 Pod。

   由于 cert-manager 不支持组合 CA 证书，若尝试滚动升级各组件，则使用新旧不同 CA 签发证书的 Pod 间将无法基于 TLS 正常通信。因此需要强制删除 Pod 并通过基于新 CA 签发的证书重建 Pod。

```
kubectl delete -n ${namespace} pod ${pod_name}
```

   其中 `${namespace}` 为 TiDB 集群部署的命名空间，`${pod_name}` 为 PD、TiKV 与 TiDB 各 replica 的 Pod 名称。

5.4.2.2 仅更新和替换组件间证书

1. 由 cert-manager 在证书过期之前自动更新各组件的证书及 Kubernetes Secret 对象。

   对于 PD、TiKV 及 TiDB 组件，在 TiDB 集群部署的命名空间下包含以下 Kubernetes Secret 对象：

   ```
   ${cluster_name}-pd-cluster-secret
   ${cluster_name}-tikv-cluster-secret
   ${cluster_name}-tidb-cluster-secret
   ${cluster_name}-cluster-client-secret
   ```

   其中 `${cluster_name}` 为集群的名字。

   若要手动更新组件间证书，可直接删除相应的 Kubernetes Secret 对象后触发 cert-manager 重新生成组件间证书。

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 证书及组件间证书进行处理。
- 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。

6 运维

6.1 滚动升级 Kubernetes 上的 TiDB 集群

滚动更新 TiDB 集群时，会按 PD、TiKV、TiDB 的顺序，串行删除 Pod，并创建新版本的 Pod，当新版本的 Pod 正常运行后，再处理下一个 Pod。

滚动升级过程会自动处理 PD、TiKV 的 Leader 迁移与 TiDB 的 DDL Owner 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。对于无法进行重试的客户端，滚动更新 TiDB 则会导致连接到被关闭节点的数据库连接失效，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在低峰期进行 TiDB 的滚动升级操作。

滚动更新可以用于升级 TiDB 版本，也可以用于更新集群配置。

6.1.1 通过 TidbCluster CR 升级

如果 TiDB 集群是直接通过 TidbCluster CR 部署的或者通过 Helm 方式部署的然后又切换到了 TidbCluster CR 管理，可以通过下面步骤升级 TiDB 集群。

6.1.1.1 升级 TiDB 版本

1. 修改集群的 TidbCluster CR 中各组件的镜像配置。

   正常情况下，集群内的各组件应该使用相同版本，所以一般修改 `spec.version` 即可，如果要为集群内不同组件设置不同的版本，可以修改 `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`。

   `version` 字段格式如下：

   - `spec.version`，格式为 `imageTag`，例如 `v5.0.6`
   - `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`，格式为 `imageTag`，例如 `v3.1.0`

   ```
   kubectl edit tc ${cluster_name} -n ${namespace}
   ```

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。

6.1.1.2 强制升级 TiDB 集群

如果 PD 集群因为 PD 配置错误、PD 镜像 tag 错误、NodeAffinity 等原因不可用，TiDB 集群扩缩容、升级 TiDB 版本和更新 TiDB 集群配置这三种操作都无法成功执行。

这种情况下，可使用 `force-upgrade` 强制升级集群以恢复集群功能。首先为集群设置 `annotation`：

```
kubectl annotate --overwrite tc ${cluster_name} -n ${namespace} tidb.pingcap
    ↪ .com/force-upgrade=true
```

然后修改 PD 相关配置，确保 PD 进入正常状态。

> 警告：
>
> PD 集群恢复后，必须执行下面命令禁用强制升级功能，否则下次升级过程可能会出现异常：
>
> ```
> kubectl annotate tc ${cluster_name} -n ${namespace} tidb.
>     ↪ pingcap.com/force-upgrade-
> ```

6.1.1.3 修改 TiDB 集群配置

> 注意：
>
> - 在首次启动成功后，PD 部分配置项会持久化到 etcd 中，且后续将以 etcd 中的配置为准。因此在 PD 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。
> - 通过在线修改集群配置进行修改的配置项，在滚动升级后可能被 CR 中的配置项覆盖。

1. 参考配置 TiDB 组件修改集群的 TidbCluster CR 中各组件配置。

```
kubectl edit tc ${cluster_name} -n ${namespace}
```

2. 查看配置修改后的更新进度：

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

当所有 Pod 都重建完毕进入 `Running` 状态后，配置修改完成。

> 注意：
>
> TiDB（v4.0.2 起）默认会定期收集使用情况信息，并将这些信息分享给 PingCAP 用于改善产品。若要了解所收集的信息详情及如何禁用该行为，请参见遥测。

6.2 升级 TiDB Operator

本文介绍如何升级 TiDB Operator。

> 注意：
>
> 你可以通过 `helm search repo -l tidb-operator` 命令查看当前支持的 TiDB Operator 版本。如果此命令的输出中未包含最新版本，可以使用 `helm repo update` 命令更新 repo。详情请参考配置 Helm repo 。

6.2.1 在线升级步骤

1. 更新 Kubernetes 的 CustomResourceDefinition (CRD)。关于 CRD 的更多信息，请参阅 CustomResourceDefinition。

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/manifests/crd.yaml && \
kubectl get crd tidbclusters.pingcap.com
```

2. 获取你要升级的 tidb-operator chart 中的 values.yaml 文件：

```
mkdir -p ${HOME}/tidb-operator/v1.1.15 && \
helm inspect values pingcap/tidb-operator --version=v1.1.15 > ${HOME}/
    ↪ tidb-operator/v1.1.15/values-tidb-operator.yaml -n tidb-admin
```

3. 修改 ${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml 中 operatorImage ↪ 镜像版本为要升级到的版本，并将旧版本 values.yaml 中的自定义配置合并到 ${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml，然后执行 helm upgrade：

```
helm upgrade tidb-operator pingcap/tidb-operator --version=v1.1.15 -f $
    ↪ {HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml
```

Pod 全部正常启动之后，运行以下命令确认 TiDB Operator 镜像版本：

```
kubectl get po -n tidb-admin -l app.kubernetes.io/instance=tidb-
    ↪ operator -o yaml | grep 'image:.*operator:'
```

输出类似下方结果则表示升级成功，v1.1.15表示要升级到的版本号。

```
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
```

> **注意：**
>
> TiDB Operator 升级之后，所有 TiDB 集群中的 discovery deployment 都会自动升级到对应的 TiDB Operator 版本。

6.2.2 离线升级步骤

如果服务器没有连接外网，你可以按照以下步骤离线升级 TiDB Operator：

1. 使用有外网的机器下载升级所需的文件和镜像。

1. 下载 TiDB Operator 需要的 crd.yaml 文件。关于 CRD 的更多信息，请参阅 CustomResourceDefinition。

```
wget https://raw.githubusercontent.com/pingcap/tidb-operator/v1
    ↪ .1.15/manifests/crd.yaml
```

2. 下载 tidb-operator chart 包文件：

```
wget http://charts.pingcap.org/tidb-operator-v1.1.15.tgz
```

3. 下载 TiDB Operator 升级所需的 Docker 镜像:

```
docker pull pingcap/tidb-operator:v1.1.15
docker pull pingcap/tidb-backup-manager:v1.1.15

docker save -o tidb-operator-v1.1.15.tar pingcap/tidb-operator:v1
    ↪ .1.15
docker save -o tidb-backup-manager-v1.1.15.tar pingcap/tidb-backup-
    ↪ manager:v1.1.15
```

2. 将下载的文件和镜像上传到需要升级的服务器上，然后按照以下步骤进行安装：

1. 安装 TiDB Operator 需要的 crd.yaml 文件：

```
kubectl apply -f ./crd.yaml
```

2. 解压 tidb-operator chart 包文件，并拷贝 values.yaml 文件：

```
tar zxvf tidb-operator-v1.1.15.tgz && \
mkdir -p ${HOME}/tidb-operator/v1.1.15 && \
cp tidb-operator/values.yaml ${HOME}/tidb-operator/v1.1.15/values-
    ↪ tidb-operator.yaml
```

3. 安装 Docker 镜像到服务器上：

```
docker load -i tidb-operator-v1.1.15.tar
docker load -i tidb-backup-manager-v1.1.15.tar
```

3. 修改 ${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml 中 operatorImage ↪ 镜像版本为要升级到的版本，并将旧版本 values.yaml 中的自定义配置合并到 ${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml，然后执行 helm upgrade：

```
helm upgrade tidb-operator ./tidb-operator --version=v1.1.15 -f ${HOME
    ↪ }/tidb-operator/v1.1.15/values-tidb-operator.yaml
```

Pod 全部正常启动之后，运行以下命令确认 TiDB Operator 镜像版本：

```shell
kubectl get po -n tidb-admin -l app.kubernetes.io/instance=tidb-operator -o
    ↪ yaml | grep 'image:.*operator:'
```

输出类似下方结果则表示升级成功，v1.1.15表示要升级到的版本号。

```
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
```

注意：

TiDB Operator 升级之后，所有 TiDB 集群中的 discovery deployment 都会自动升级到对应的 TiDB Operator 版本。

6.2.3 从 TiDB Operator v1.0 版本升级到 v1.1 及之后版本

从 TiDB Operator v1.1.0 开始，PingCAP 不再继续更新维护 tidb-cluster chart，原来由 tidb-cluster chart 负责管理的组件或者功能在 v1.1 中切换到 CR (Custom Resource) 或者单独的 chart 进行管理，详细信息请参考TiDB Operator v1.1 重要注意事项。

6.3 TiDB Operator 灰度升级

本文介绍如何灰度升级 TiDB Operator。灰度升级可以控制 TiDB Operator 升级的影响范围，避免由于 TiDB Operator 升级导致对整个 Kubernetes 集群中的所有 TiDB 集群产生不可预知的影响，在确认 TiDB Operator 升级的影响或者确认 TiDB Operator 新版本能正常稳定工作后再正常升级 TiDB Operator。

注意：

- 目前仅支持灰度升级 tidb-controller-manager 和 tidb-scheduler，不支持灰度升级 AdvancedStatefulSet controller 和 AdmissionWebhook。
- v1.1.10 开始支持此项功能，所以当前 TiDB Operator 版本需 >= v1.1.10。

6.3.1 相关参数

为了支持灰度升级 TiDB Operator，tidb-operator chart 中 values.yaml 文件里面添加了一些参数，可以参考文档。

6.3.2 灰度升级 TiDB Operator

1. 为当前 TiDB Operator 配置 selector。

   参考升级 TiDB Operator 文档，在 values.yaml 中添加如下配置，升级当前 TiDB Operator：

   ```
   controllerManager:
     selector:
     - version!=canary
   ```

   如果之前已经执行过上述步骤，可以直接进入下一步。

2. 部署灰度 TiDB Operator。

   参考部署 TiDB Operator 文档，在 values.yaml 中添加如下配置，在不同的 namespace 中（例如 tidb-admin-canary）使用不同的 **Helm Release Name**（例如 helm install tidb-operator-canary ...）部署灰度 TiDB Operator：

   ```
   controllerManager:
     selector:
     - version=canary
   appendReleaseSuffix: true
   #scheduler:
   #  create: false
   advancedStatefulset:
     create: false
   admissionWebhook:
     create: false
   ```

> 注意：
>
> - 建议在单独的 namespace 部署新的 TiDB Operator。
> - appendReleaseSuffix 需要设置为 true。
> - 如果不需要灰度升级 tidb-scheduler，可以设置 scheduler.create: false。
> - 如果配置 scheduler.create: true，会创建一个名字为 {{.scheduler.schedulerName}}-{{.Release.Name}} 的 scheduler，要使用这个 scheduler，需要配置 TidbCluster CR 中的 spec.schedulerName 为这个 scheduler。

- 由于不支持灰度升级 AdvancedStatefulSet controller 和 AdmissionWebhook，需要配置 `advancedStatefulset.create: false` 和 `admissionWebhook.create: false`。

3. 如果需要测试 tidb-controller-manager 灰度升级，通过如下命令为某个 TiDB 集群设置 label：

```
kubectl -n ${namespace} label tc ${cluster_name} version=canary
```

通过查看已经部署的两个 tidb-controller-manager 的日志可以确认，这个 TiDB 集群已经归灰度 TiDB Operator 管理。

1. 查看当前 TiDB Operator `tidb-controller-manager` 的日志:

```
kubectl -n tidb-admin logs tidb-controller-manager-55b887bdc9-lzdwv
```

```
I0305 07:52:04.558973    1 tidb_cluster_controller.go:148]
    ↪ TidbCluster has been deleted tidb-cluster-1/basic1
```

2. 查看灰度 TiDB Operator `tidb-controller-manager` 的日志:

```
kubectl -n tidb-admin-canary logs tidb-controller-manager-canary-6
    ↪ dcb9bdd95-qf4qr
```

```
I0113 03:38:43.859387    1 tidbcluster_control.go:69] TidbCluster:
    ↪ [tidb-cluster-1/basic1] updated successfully
```

4. 如果需要测试 tidb-scheduler 灰度升级，通过如下命令为某个 TiDB 集群修改 `spec.schedulerName` 为 `tidb-scheduler-canary`：

```
kubectl -n ${namespace} edit tc ${cluster_name}
```

修改后，集群内各组件会滚动升级，可以通过查看灰度 TiDB Operator `tidb-scheduler` 的日志确认集群已经使用灰度 `tidb-scheduler`：

```
kubectl -n tidb-admin-canary logs tidb-scheduler-canary-7f7b6c7c6-j5p2j
    ↪  -c tidb-scheduler
```

5. 灰度测试完成后，可以将 3，4 步骤中的修改改回去，重新使用当前 TiDB Operator 管理。

```
kubectl -n ${namespace} label tc ${cluster_name} version-
```

```
kubectl -n ${namespace} edit tc ${cluster_name}
```

6. 删除灰度 TiDB Operator。

```
helm -n tidb-admin-canary uninstall ${release_name}
```

7. 参考升级 TiDB Operator 文档正常升级当前 TiDB Operator。

6.4 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。

6.4.1 什么是同步

在 TiDB Operator 中，控制器会不断对比 `TidbCluster` 对象中记录的期望状态与 TiDB 集群的实际状态，并调整 Kubernetes 中的资源以驱动 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。

6.4.2 暂停同步的应用场景

以下为一些暂停同步的应用场景。

- 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。

6.4.3 暂停同步 TiDB 集群

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 edit tc ${cluster_name} -n ${namespace}
```

2. 在 TidbCluster CR 中以如下方式配置 `spec.paused: true`，保存配置并退出编辑器。TiDB 集群各组件 (PD、TiKV、TiDB、TiFlash、TiCDC、Pump) 的同步过程将会被暂停。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  ...
spec:
  ...
  paused: true # 暂停同步
  pd:
    ...
  tikv:
    ...
  tidb:
    ...
```

3. TiDB 集群同步暂停后，可以使用以下命令查看 tidb-controller-manager Pod 日志确认 TiDB 集群同步状态。其中 `${pod_name}` 表示 tidb-controller-manager Pod 的名称，`${namespace}` 表示 TiDB Operator 所在的 namespace。

```
kubectl logs ${pod_name} -n ${namespace} | grep paused
```

输出类似下方结果则表示 TiDB 集群同步已经暂停。

```
I1207 11:09:59.029949    1 pd_member_manager.go:92] tidb cluster
   ↪ default/basic is paused, skip syncing for pd service
I1207 11:09:59.029977    1 pd_member_manager.go:136] tidb cluster
   ↪ default/basic is paused, skip syncing for pd headless service
I1207 11:09:59.035437    1 pd_member_manager.go:191] tidb cluster
   ↪ default/basic is paused, skip syncing for pd statefulset
I1207 11:09:59.035462    1 tikv_member_manager.go:116] tikv cluster
   ↪ default/basic is paused, skip syncing for tikv service
I1207 11:09:59.036855    1 tikv_member_manager.go:175] tikv cluster
   ↪ default/basic is paused, skip syncing for tikv statefulset
I1207 11:09:59.036886    1 tidb_member_manager.go:132] tidb cluster
   ↪ default/basic is paused, skip syncing for tidb headless service
I1207 11:09:59.036895    1 tidb_member_manager.go:258] tidb cluster
   ↪ default/basic is paused, skip syncing for tidb service
I1207 11:09:59.039358    1 tidb_member_manager.go:188] tidb cluster
   ↪ default/basic is paused, skip syncing for tidb statefulset
```

6.4.4 恢复同步 TiDB 集群

如果想要恢复 TiDB 集群的同步，可以在 TidbCluster CR 中配置 `spec.paused: false`，恢复同步 TiDB 集群。

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 edit tc ${cluster_name} -n ${namespace}
```

2. 在 TidbCluster CR 中以如下方式配置 `spec.paused: false`，保存配置并退出编辑器。TiDB 集群各组件 (PD、TiKV、TiDB、TiFlash、TiCDC、Pump) 的同步过程将会被恢复。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  ...
spec:
  ...
  paused: false # 恢复同步
  pd:
    ...
  tikv:
    ...
  tidb:
    ...
```

3. 恢复 TiDB 集群同步后，可以使用以下命令查看 tidb-controller-manager Pod 日志确认 TiDB 集群同步状态。其中 `${pod_name}` 表示 tidb-controller-manager Pod 的名称，`${namespace}` 表示 TiDB Operator 所在的 namespace。

```
kubectl logs ${pod_name} -n ${namespace} | grep "Finished syncing
    ↪ TidbCluster"
```

输出类似下方结果，可以看到同步成功时间戳大于暂停同步日志中显示的时间戳，表示 TiDB 集群同步已经被恢复。

```
I1207 11:14:59.361353    1 tidb_cluster_controller.go:136] Finished
    ↪ syncing TidbCluster "default/basic" (368.816685ms)
I1207 11:15:28.982910    1 tidb_cluster_controller.go:136] Finished
    ↪ syncing TidbCluster "default/basic" (97.486818ms)
I1207 11:15:29.360446    1 tidb_cluster_controller.go:136] Finished
    ↪ syncing TidbCluster "default/basic" (377.51187ms)
```

6.5 TiDB 集群伸缩

6.5.1 Kubernetes 上的 TiDB 集群扩缩容

本文介绍 TiDB 在 Kubernetes 中如何进行水平扩缩容和垂直扩缩容。

6.5.1.1 水平扩缩容

TiDB 水平扩缩容操作指的是通过增加或减少节点的数量，来达到集群扩缩容的目的。扩缩容 TiDB 集群时，会按照填入的 replicas 值，对 PD、TiKV、TiDB 进行顺序扩缩容操作。扩容操作按照节点编号由小到大增加节点，缩容操作按照节点编号由大到小删除节点。目前 TiDB 集群使用 TidbCluster Custom Resource (CR) 管理方式。

6.5.1.1.1 扩缩容 PD、TiDB、TiKV

使用 kubectl 修改集群所对应的 `TidbCluster` 对象中的 `spec.pd.replicas`、`spec.tidb.replicas`、`spec.tikv.replicas` 至期望值。

同样，你可以使用以下命令在线修改 Kubernetes 集群中的 `TidbCluster` 定义。

```
kubectl edit tidbcluster ${cluster_name} -n ${namespace}
```

你可以通过以下指令查看 Kubernetes 集群中对应的 TiDB 集群是否更新到了你的期望定义。

```
kubectl get tidbcluster ${cluster_name} -n ${namespace} -oyaml
```

如果上述指令输出的 `TidbCluster` 中，`spec.pd.replicas`、`spec.tidb.replicas`、`spec.tikv.replicas` 的值和你之前更新的值一致，那么可以通过以下指令来观察 `TidbCluster` Pod 是否新增或者减少。对于 PD 和 TiDB 而言，会需要 10 到 30 秒左右的时间进行扩容或者缩容。对于 TiKV 组件，由于涉及到数据搬迁，可能会需要 3 到 5 分钟来进行扩容或者缩容。

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

扩容 TiFlash

如果集群中部署了 TiFlash，可以通过修改 `spec.tiflash.replicas` 对 TiFlash 进行扩容。

扩缩容 TiCDC

如果集群中部署了 TiCDC，可以通过修改 `spec.ticdc.replicas` 对 TiCDC 进行扩缩容。

缩容 TiFlash

1. 通过 `port-forward` 暴露 PD 服务：

   ```
   kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379
   ```

2. 打开一个新终端标签或窗口，通过如下命令确认开启 TiFlash 的所有数据表的最大副本数 N：

```
curl 127.0.0.1:2379/pd/api/v1/config/rules/group/tiflash | grep count
```

输出结果中 count 的最大值就是所有数据表的最大副本数 N。

3. 回到 port-forward 命令所在窗口，按 Ctrl+C 停止 port-forward。

4. 如果缩容 TiFlash 后，TiFlash 集群剩余 Pod 数大于等于所有数据表的最大副本数 N，直接进行下面第 6 步。如果缩容 TiFlash 后，TiFlash 集群剩余 Pod 数小于所有数据表的最大副本数 N，参考访问 TiDB 集群的步骤连接到 TiDB 服务，并针对所有副本数大于集群剩余 TiFlash Pod 数的表执行如下命令：

```
alter table <db_name>.<table_name> set tiflash replica 0;
```

5. 等待相关表的 TiFlash 副本被删除。

连接到 TiDB 服务，执行如下命令，查不到相关表的同步信息时即为副本被删除：

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA =
    ↪ '<db_name>' and TABLE_NAME = '<table_name>';
```

6. 修改 spec.tiflash.replicas 对 TiFlash 进行缩容。

你可以通过以下指令查看 Kubernetes 集群中对应的 TiDB 集群中的 TiFlash 是否更新到了你的期望定义。检查以下指令输出内容中，spec.tiflash.replicas 的值是否符合预期值。

```
kubectl get tidbcluster ${cluster-name} -n ${namespace} -oyaml
```

6.5.1.1.2 查看集群水平扩缩容状态

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

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。

注意：

- PD、TiKV、TiFlash 组件在扩缩容的过程中不会触发滚动升级操作。
- 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} tidbcluster ${cluster_name} -o json | jq '.status.tikv.stores' 查看 TiKV 是否处于下线 Offline 状态。

- 当 TiKV UP 状态的 store 数量 <= 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。假如异常已经发生，可以参考TiKV Store 异常进入 Tombstone 状态 进行解决。
- TiFlash 组件缩容处理逻辑和 TiKV 组件相同。
- PD、TiKV、TiFlash 组件在缩容过程中被删除的节点的 PVC 会保留，并且由于 PV 的 Reclaim Policy 设置为 Retain，即使 PVC 被删除，数据依然可以找回。

6.5.1.1.3 水平扩缩容故障

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。

6.5.1.2 垂直扩缩容

垂直扩缩容操作指的是通过增加或减少节点的资源限制，来达到集群扩缩容的目的。垂直扩缩容本质上是节点滚动升级的过程。目前 TiDB 集群使用 TidbCluster Custom Resource (CR) 管理方式。

6.5.1.2.1 垂直扩缩容操作

通过 kubectl 修改集群所对应的 TidbCluster 对象的 spec.pd.resources、spec.tikv.resources、spec.tidb.resources 至期望值。

如果集群中部署了 TiFlash，可以通过修改 spec.tiflash.resources 对 TiFlash 进行垂直扩缩容。

如果集群中部署了 TiCDC，可以通过修改 spec.ticdc.resources 对 TiCDC 进行垂直扩缩容。

6.5.1.2.2 查看垂直扩缩容进度

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

当所有 Pod 都重建完毕进入 Running 状态后，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 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 作为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。

6.5.1.3 垂直扩缩容故障

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。

6.5.2 启用 TidbCluster 弹性伸缩

在 Kubernetes 平台上，有着基于 CPU 利用率进行负载的原生 API: Horizontal Pod Autoscaler。基于 Kubernetes，TiDB 4.0 起支持了全新的弹性调度算法。与之相应的，在 TiDB Operator 1.1 及以上版本中，TiDB 集群可以凭借 Kubernetes 平台本身的特性来开启弹性调度的能力。本篇文章将会介绍如何开启并使用 TidbCluster 的弹性伸缩能力。

6.5.2.1 开启弹性伸缩特性

警告：

- TidbCluster 弹性伸缩目前仍处于 Alpha 阶段，我们极其不推荐在关键、生产环境开启这个特性
- 我们推荐你在测试、内网环境对这个特性进行体验，并反馈相关的建议与问题给我们，帮助我们更好地提高这一特性能力。

开启弹性伸缩特性需要主动开启 Operator 相关配置，默认情况下 Operator 的弹性伸缩特性是关闭的。你可以通过以下方式来开启弹性调度特性:

1. 修改 Operator 的 `values.yaml`

   在 `features` 选项中开启 AutoScaling：

```
features:
  - AutoScaling=true
```

   开启 Operator Webhook 特性:

```
admissionWebhook:
  create: true
  mutation:
    pods: true
```

关于 Operator Webhook，请参考开启 TiDB Operator 准入控制器

2. 安装/更新 TiDB Operator

   修改完 values.yaml 文件中上述配置项以后进行 TiDB Operator 部署或者更新。安装与更新 Operator 请参考在 Kubernetes 上部署 TiDB Operator。

3. 确认目标 TiDB 集群资源设置

   目标 TiDB 集群在使用弹性伸缩前，首先需要设置好对应组件的 CPU 设置。以 TiKV 为例，你需要申明 spec.tikv.requests.cpu：

```
spec:
  tikv:
    requests:
      cpu: "1"
  tidb:
    requests:
      cpu: "1"
```

6.5.2.2 了解 TidbClusterAutoScaler

我们通过 TidbClusterAutoScaler CR 对象来控制 TiDB 集群的弹性伸缩行为，如果你曾经使用过 Horizontal Pod Autoscaler，那么你一定会对这个概念感到非常熟悉。以下是一个 TiKV 的弹性伸缩例子。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaling-demo
spec:
  cluster:
    name: auto-scaling-demo
    namespace: default
  monitor:
    name: auto-scaling-demo
    namespace: default
  tikv:
    minReplicas: 3
    maxReplicas: 4
    metrics:
      - type: "Resource"
```

```
    resource:
      name: "cpu"
      target:
        type: "Utilization"
        averageUtilization: 80
```

对于 TiDB 组件，你可以通过 spec.tidb 来进行配置，目前 TiKV 与 TiDB 的弹性伸缩 API 相同。

在 TidbClusterAutoScaler 对象中，cluster 属性代表了需要被弹性调度的 TiDB 集群，通过 name 与 namespace 来标识。由于 TidbClusterAutoScaler 组件需要通过指标采集组件抓取相关资源使用情况，我们需要提供对应的指标采集与查询服务给 TidbClusterAutoScaler。monitor 属性则代表了与之相关连的 TidbMonitor 对象。如果你不了解 TidbMonitor，可以参考TiDB 集群监控与告警。

对于非 TidbMonitor 的外部 Prometheus, 你可以通过 spec.metricsUrl 来填写这个服务的 Host，从而指定该 TiDB 集群的监控指标采集服务。对于使用 Helm 部署 TiDB 集群监控的情况，可以通过以下方式来指定 spec.metricsUrl。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaling-demo
spec:
  cluster:
    name: auto-scaling-demo
    namespace: default
  metricsUrl: "http://${release_name}-prometheus.${namespace}.svc:9090"
  ......
```

6.5.2.3 例子

1. 执行以下命令在 Kubernetes 集群上快速安装一个 1 PD、3 TiKV、2 TiDB，并带有监控与弹性伸缩能力的 TiDB 集群。

```
$ kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/examples/auto-scale/tidb-cluster.yaml -n ${
    ↪ namespace}
```

```
$ kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/examples/auto-scale/tidb-monitor.yaml -n ${
    ↪ namespace}
```

```
$ kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/examples/auto-scale/tidb-cluster-auto-scaler.
    ↪ yaml -n ${namespace}
```

2. 当 TiDB 集群创建完毕以后，使用以下方式暴露 TiDB 集群服务到本地。

```
kubectl port-forward svc/auto-scaling-demo-tidb 4000:4000 &
```

将以下内容复制到本地的 sysbench.config 文件中：

```
mysql-host=127.0.0.1
mysql-port=4000
mysql-user=root
mysql-password=
mysql-db=test
time=120
threads=20
report-interval=5
db-driver=mysql
```

3. 使用 sysbench 工具准备数据并进行压测。

将以下内容复制到本地的 sysbench.config 文件中：

```
mysql-host=127.0.0.1
mysql-port=4000
mysql-user=root
mysql-password=
mysql-db=test
time=120
threads=20
report-interval=5
db-driver=mysql
```

通过以下命令准备数据：

```
sysbench --config-file=${path-to-file}/sysbench.config
    ↪ oltp_point_select --tables=1 --table-size=20000 prepare
```

通过以下命令开始进行压测：

```
sysbench --config-file=${path-to-file}/sysbench.config
    ↪ oltp_point_select --tables=1 --table-size=20000 run
```

上述命令执行完毕后，出现如下输出：

```
Initializing worker threads...

Threads started!

[ 5s ] thds: 20 tps: 37686.35 qps: 37686.35 (r/w/o: 37686.35/0.00/0.00)
    ↪  lat (ms,95%): 0.99 err/s: 0.00 reconn/s: 0.00
```

```
[ 10s ] thds: 20 tps: 38487.20 qps: 38487.20 (r/w/o:
    ↪ 38487.20/0.00/0.00) lat (ms,95%): 0.95 err/s: 0.00 reconn/s: 0.00
```

4. 新建一个会话终端，通过以下命令观察 TiDB 集群的 Pod 变化情况。

```
watch -n1 "kubectl -n ${namespace} get pod"
```

出现如下输出:

```
auto-scaling-demo-discovery-fbd95b679-f4cb9 1/1 Running 0          17m
auto-scaling-demo-monitor-6857c58564-ftkp4 3/3 Running  0          17m
auto-scaling-demo-pd-0                      1/1    Running  0          17m
auto-scaling-demo-tidb-0                    2/2    Running  0          15m
auto-scaling-demo-tidb-1                    2/2    Running  0          15m
auto-scaling-demo-tikv-0                    1/1    Running  0          15m
auto-scaling-demo-tikv-1                    1/1    Running  0          15m
auto-scaling-demo-tikv-2                    1/1    Running  0          15m
```

观察 Pod 的变化情况与 sysbench 的 TPS 与 QPS，当 TiKV 与 TiDB Pod 新增时，sysbench 的 TPS 与 QPS 值有显著提升。当 sysbench 结束后，观察 Pod 变化情况，发现新增的 TiKV 与 TiDB Pod 自动消失。

5. 使用如下命令销毁环境：

```
kubectl delete tidbcluster auto-scaling-demo -n ${namespace}
kubectl delete tidbmonitor auto-scaling-demo -n ${namespace}
kubectl delete tidbclusterautoscaler auto-scaling-demo -n ${namespace}
```

6.5.2.4 配置 TidbClusterAutoScaler

1. 设置弹性伸缩间隔

相比无状态的 Web 服务，一个分布式数据库软件对于实例的伸缩往往是非常敏感的。我们需要保证每次弹性伸缩之间存在一定的间隔，从而避免引起频繁的弹性伸缩。

你可以通过 `spec.tikv.scaleInIntervalSeconds` 和 `spec.tikv.ScaleOutIntervalSeconds` ↪ 来配置每两次弹性伸缩之间的时间间隔 (秒)，对于 TiDB 也同样如此。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-sclaer
spec:
  tidb:
    scaleInIntervalSeconds: 500
    ScaleOutIntervalSeconds: 300
  tikv:
```

```
    scaleInIntervalSeconds: 500
    ScaleOutIntervalSeconds: 300
```

2. 设置最大最小值

   就像 Horizontal Pod Autoscaler，在 TidbClusterAutoScaler 中你也可以设置给每个组件最大最小值来控制 TiDB、TiKV 的伸缩范围。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaling-demo
spec:
  tikv:
    minReplicas: 3
    maxReplicas: 4
  tidb:
    minReplicas: 2
    maxReplicas: 3
```

3. 配置 CPU 弹性伸缩

   目前 TidbClusterAutoScaler 仅支持基于 CPU 负载的弹性伸缩，CPU 负载的描述性 API 如下所示。averageUtilization 则代表了 CPU 负载利用率的阈值。如果当前 CPU 利用率超过 80%，则会触发弹性扩容。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaling-demo
spec:
  tikv:
    minReplicas: 3
    maxReplicas: 4
    metrics:
      - type: "Resource"
        resource:
          name: "cpu"
          target:
            type: "Utilization"
            averageUtilization: 80
```

4. 配置指标时间窗口

   目前基于 CPU 负载的弹性调度，TidbClusterAutoScaler 会在所指定的监控系统中获取 TiDB、TiKV 的 CPU 指标，你可以指定采集指标的时间窗口。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: basic
  tidb:
    metricsTimeDuration: "1m"
    metrics:
      - type: "Resource"
        resource:
          name: "cpu"
          target:
            type: "Utilization"
            averageUtilization: 60
```

6.6 备份与恢复

6.6.1 备份与恢复简介

本文档介绍如何使用 BR、Dumpling、TiDB Lightning 对 Kubernetes 上的 TiDB 集群进行数据备份和数据恢复。

TiDB Operator 1.1 及以上版本推荐使用基于 CustomResourceDefinition (CRD) 实现的备份恢复方式实现：

- 如果 TiDB 集群版本 >= v3.1，可以参考以下文档：
  - 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 集群到持久卷
  - 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 恢复持久卷上的备份数据
- 如果 TiDB 集群版本 < v3.1，可以参考以下文档：
  - 使用 Dumpling 备份 TiDB 集群数据到兼容 S3 的存储
  - 使用 Dumpling 备份 TiDB 集群数据到 GCS
  - 使用 TiDB Lightning 恢复兼容 S3 的存储上的备份数据
  - 使用 TiDB Lightning 恢复 GCS 上的备份数据

6.6.1.1 使用场景

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。如果需要直接备份 SST

文件（键值对）或者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。

TiDB Lightning 是一个将全量数据高速导入到 TiDB 集群的工具，TiDB Lightning 有以下两个主要的使用场景：一是大量新数据的快速导入；二是全量备份数据的恢复。目前，TiDB Lightning 支持 Dumpling 或 CSV 输出格式的数据源。你可以在以下两种场景下使用 TiDB Lightning：

- 迅速导入大量新数据。
- 恢复所有备份数据。

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。

6.6.1.2 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 Custom Resource (CR) 对象来描述一次备份，具体备份过程可参考备份与恢复简介中列出的文档。以下介绍 Backup CR 各个字段的具体含义。

6.6.1.2.1 通用字段介绍

- `.spec.metadata.namespace`：Backup CR 所在的 namespace。
- `.spec.toolImage`：用于指定 Backup 使用的工具镜像。
  - 使用 BR 备份时，可以用该字段指定 BR 的版本：
    * 如果未指定或者为空，默认使用镜像 `pingcap/br:${tikv_version}` 进行备份。
    * 如果指定了 BR 的版本，例如 `.spec.toolImage: pingcap/br:v5.0.6`，那么使用指定的版本镜像进行备份。
    * 如果指定了镜像但未指定版本，例如 `.spec.toolImage: private/registry/br`，那么使用镜像 `private/registry/br:${tikv_version}` 进行备份。
  - 使用 Dumpling 备份时，可以用该字段指定 Dumpling 的版本，例如，`spec.toolImage: pingcap/dumpling:v5.0.6`。如果不指定，默认使用 Backup Manager Dockerfile 文件中 `TOOLKIT_V40` 指定的 Dumpling 版本进行备份。
  - TiDB Operator 从 v1.1.9 版本起支持这项配置。
- `.spec.tikvGCLifeTime`：备份中的临时 `tikv_gc_life_time` 时间设置，默认为 72h。在备份开始之前，若 TiDB 集群的 `tikv_gc_life_time` 小于用户设置的 `spec.tikvGCLifeTime`，为了保证备份的数据不被 TiKV GC 掉，TiDB Operator 会在备份前调节 `tikv_gc_life_time` 为 `spec.tikvGCLifeTime`。

备份结束后不论成功或者失败，只要老的 tikv_gc_life_time 比设置的 .spec.tikvGCLifeTime 小，TiDB Operator 都会尝试恢复 tikv_gc_life_time 为备份前的值。在极端情况下，TiDB Operator 访问数据库失败会导致 TiDB Operator 无法自动恢复 tikv_gc_life_time 并认为备份失败。

此时，可以通过下述语句查看当前 TiDB 集群的 tikv_gc_life_time：

```
select VARIABLE_NAME, VARIABLE_VALUE from mysql.tidb where
    ↪ VARIABLE_NAME like "tikv_gc_life_time";
```

如果发现 tikv_gc_life_time 值过大 (通常为 10m),则需要按照调节 tikv_gc_life_time 将 tikv_gc_life_time 调回原样。

- .spec.cleanPolicy：备份集群后删除备份 CR 时的备份文件清理策略。目前支持三种清理策略：
  - Retain：任何情况下，删除备份 CR 时会保留备份出的文件
  - Delete：任何情况下，删除备份 CR 时会删除备份出的文件
  - OnFailure：如果备份中失败，删除备份 CR 时会删除备份出的文件

  如果不配置该字段，或者配置该字段的值为上述三种以外的值，均会保留备份出的文件。值得注意的是，在 v1.1.2 以及之前版本不存在该字段，且默认在删除 CR 的同时删除备份的文件。若 v1.1.3 及之后版本的用户希望保持该行为，需要设置该字段为 Delete。
- .spec.from.host：待备份 TiDB 集群的访问地址，为需要导出的 TiDB 的 service name，例如 basic-tidb。
- .spec.from.port：待备份 TiDB 集群的访问端口。
- .spec.from.user：待备份 TiDB 集群的访问用户。
- .spec.from.secretName：存储 .spec.from.user 用户的密码的 Secret。
- .spec.from.tlsClientSecretName：指定备份使用的存储证书的 Secret。

  如果 TiDB 集群开启了TLS，但是不想使用文档中创建的 ${cluster_name}-cluster-client-secret 进行备份，可以通过这个参数为备份指定一个 Secret，可以通过如下命令生成：

```
kubectl create secret generic ${secret_name} --namespace=${namespace}
    ↪ --from-file=tls.crt=${cert_path} --from-file=tls.key=${key_path}
    ↪ --from-file=ca.crt=${ca_path}
```

- .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 Gi。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.tableFilter`：备份时指定让 Dumpling 或者 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，如果使用 Dumpling 备份，`tableFilter` 字段的默认值如下：

```
tableFilter:
- "*.*"
- "!/^(mysql|test|INFORMATION_SCHEMA|PERFORMANCE_SCHEMA|METRICS_SCHEMA|
    ↪ INSPECTION_SCHEMA)$/.*"
```

  如果使用 BR 备份，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"
> ```

6.6.1.2.2 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 权限或者 GCP 权限传输给 TiKV 进程。默认为 `true`。
- `.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。自 v1.1.6 版本起支持该参数。可用于指定 `lastbackupts` 以进行增量备份。

6.6.1.2.3 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.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。这几种访问控制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`。这几种存储类型的详细介绍参考 AWS 官方文档。

6.6.1.2.4 GCS 存储字段介绍

- `.spec.gcs.projectId`：代表 GCP 上用户项目的唯一标识。具体获取该标识的方法可参考 GCP 官方文档。
- `.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。这几种访问控制策略的详细介绍可参考 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`。这几种访问控制策略的详细介绍可参考 GCS 官方文档。

6.6.1.2.5 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。

6.6.1.3 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` Custom Resource (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.metadata.namespace`：`Restore` CR 所在的 namespace。
- `.spec.toolImage`：用于指定 `Restore` 使用的工具镜像。
  - 使用 BR 恢复时，可以用该字段指定 BR 的版本。例如，`spec.toolImage: pingcap/br:v4.0.10`。如果不指定，默认使用 `pingcap/br:${tikv_version}` 进行恢复。
  - 使用 Lightning 恢复时，可以用该字段指定 Lightning 的版本，例如`spec.toolImage: pingcap/lightning:v4.0.10`。如果不指定，默认使用 Backup Manager Dockerfile 文件中 `TOOLKIT_V40` 指定的 Lightning 版本进行恢复。
  - TiDB Operator 从 v1.1.9 版本起支持这项配置。
- `.spec.to.host`：待恢复 TiDB 集群的访问地址。
- `.spec.to.port`：待恢复 TiDB 集群的访问端口。
- `.spec.to.user`：待恢复 TiDB 集群的访问用户。
- `.spec.to.secretName`：存储 `.spec.to.user` 用户的密码的 secret。
- `.spec.to.tlsClientSecretName`：指定恢复备份使用的存储证书的 Secret。

  如果 TiDB 集群开启了TLS，但是不想使用文档中创建的 `${cluster_name}-cluster-client-secret` 恢复备份，可以通过这个参数为恢复备份指定一个 Secret，可以通过如下命令生成：

```
kubectl create secret generic ${secret_name} --namespace=${namespace}
    --from-file=tls.crt=${cert_path} --from-file=tls.key=${key_path}
    --from-file=ca.crt=${ca_path}
```

- `.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，如果使用 TiDB Lightning 恢复，`tableFilter` 字段的默认值如下：

```
tableFilter:
- "*.*"
- "!/^(mysql|test|INFORMATION_SCHEMA|PERFORMANCE_SCHEMA|METRICS_SCHEMA|
    ↪ INSPECTION_SCHEMA)$/.*"
```

  如果使用 BR 恢复，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.local`：持久卷存储相关配置，具体介绍参考Local 字段介绍。

6.6.1.4 BackupSchedule CR 字段介绍

`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`。`backupTemplate` 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。下面介绍 `backupSchedule` 独有的配置项：

- `.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`。如果同时设置最大备份保留个数和最长备份保留时间，则以最长备份保留时间为准。
- `.spec.schedule`：Cron 的时间调度格式。具体格式可参考 Cron。
- `.spec.pause`：该值默认为 `false`。如果将该值设置为 `true`，表示暂停定时调度。此时即使到了调度时间点，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。将 `true` 改为 `false` 则重新开启定时全量备份。

6.6.1.5 删除备份的 Backup CR

用户可以通过下述语句来删除对应的备份 CR 或定时全量备份 CR。

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

如果你使用 v1.1.2 及以前版本，或使用 v1.1.3 及以后版本并将 `spec.cleanPolicy` 设置为 `Delete` 时，TiDB Operator 在删除 CR 时会同时删除备份文件。

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 edit backup ${name} -n ${namespace}
```

删除 `metadata.finalizers` 配置，即可正常删除 CR。

6.6.2 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。

6.6.2.1 AWS 账号授权

在 AWS 云环境中，不同的类型的 Kubernetes 集群提供了不同的权限授予方式。本文分别介绍以下三种权限授予配置方式。

6.6.2.1.1 通过 AccessKey 和 SecretKey 授权

AWS 的客户端支持读取进程环境变量中的 `AWS_ACCESS_KEY_ID` 以及 `AWS_SECRET_ACCESS_KEY` 来获取与之相关联的用户或者角色的权限。

创建 `s3-secret` secret，在以下命令中使用 AWS 账号的 AccessKey 和 SecretKey 进行授权。该 secret 存放用于访问 S3 兼容存储的凭证。

```
kubectl create secret generic s3-secret --from-literal=access_key=xxx --from
    ↪ -literal=secret_key=yyy --namespace=test1
```

6.6.2.1.2 通过 IAM 绑定 Pod 授权

通过将用户的 IAM 角色与所运行的 Pod 资源进行绑定，使 Pod 中运行的进程获得角色所拥有的权限，这种授权方式是由 kube2iam 提供。

注意：

- 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 hostNetwork 网络模式，请确保参数 spec.tikv.hostNetwork 的值为 false。

1. 创建 IAM 角色：

   可以参考 AWS 官方文档来为账号创建一个 IAM 角色，并且通过 AWS 官方文档为 IAM 角色赋予需要的权限。由于 Backup 需要访问 AWS 的 S3 存储，所以这里给 IAM 赋予了 AmazonS3FullAccess 的权限。

2. 绑定 IAM 到 TiKV Pod：

   在使用 BR 备份的过程中，TiKV Pod 和 BR Pod 一样需要对 S3 存储进行读写操作，所以这里需要给 TiKV Pod 打上 annotation 来绑定 IAM 角色。

   ```
   kubectl edit tc demo1 -n test1
   ```

   找到 spec.tikv.annotations，增加 annotation iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user，然后退出编辑，等到 TiKV Pod 重启后，查看 Pod 是否加上了这个 annotation。

注意：

arn:aws:iam::123456789012:role/user 为步骤 1 中创建的 IAM 角色。

6.6.2.1.3 通过 IAM 绑定 ServiceAccount 授权

通过将用户的 IAM 角色与 Kubeneters 中的 serviceAccount 资源进行绑定，从而使得使用该 ServiceAccount 账号的 Pod 都拥有该角色所拥有的权限，这种授权方式由 EKS Pod Identity Webhook 服务提供。

使用该授权模式时，可以参考 AWS 官方文档创建 EKS 集群，并且部署 TiDB Operator 以及 TiDB 集群。

1. 在集群上为服务帐户启用 IAM 角色：

   可以参考 AWS 官方文档开启所在的 EKS 集群的 IAM 角色授权。

2. 创建 IAM 角色：

   可以参考 AWS 官方文档创建一个 IAM 角色，为角色赋予 AmazonS3FullAccess 的权限，并且编辑角色的 Trust relationships。

3. 绑定 IAM 到 ServiceAccount 资源上：

```
kubectl annotate sa tidb-backup-manager eks.amazonaws.com/role-arn=arn:
    ↪ aws:iam::123456789012:role/user --namespace=test1
```

4. 将 ServiceAccount 绑定到 TiKV Pod：

```
kubectl edit tc demo1 -n test1
```

   将 spec.tikv.serviceAccount 修改为 tidb-backup-manager，等到 TiKV Pod 重启后，查看 Pod 的 serviceAccountName 是否有变化。

> 注意：
>
> arn:aws:iam::123456789012:role/user 为步骤 2 中创建的 IAM 角色。

6.6.2.2 GCS 账号授权

6.6.2.2.1 通过服务账号密钥授权

创建 gcs-secret secret。该 secret 存放用于访问 GCS 的凭证。google-credentials.
↪ json 文件存放用户从 GCP console 上下载的 service account key。具体操作参考 GCP 官方文档。

```
kubectl create secret generic gcs-secret --from-file=credentials=./google-
    ↪ credentials.json -n test1
```

6.6.3 使用 S3 兼容存储备份与恢复

6.6.3.1 使用 BR 备份 TiDB 集群数据到兼容 S3 的存储

本文详细描述了如何将运行在 AWS Kubernetes 环境中的 TiDB 集群数据备份到 AWS 的存储上。BR 会在底层获取集群的逻辑备份，然后再将备份数据上传到 AWS 的存储上。

本文使用的备份方式基于 TiDB Operator v1.1 及以上版本的 Custom Resource Definition(CRD) 实现。

6.6.3.1.1 Ad-hoc 备份

Ad-hoc 备份支持全量备份与增量备份。Ad-hoc 备份通过创建一个自定义的 Backup Custom Resource (CR) 对象来描述一次备份。TiDB Operator 根据这个 Backup 对象来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。

为了更好地描述备份的使用方式，本文档提供如下备份示例。示例假设对部署在 Kubernetes test1 这个 namespace 中的 TiDB 集群 demo1 进行数据备份，下面是具体操作过程。

Ad-hoc 备份环境准备

> **注意：**
>
> 如果使用 TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR 会自动调整 tikv_gc_life_time 参数，不需要在 Backup CR 中配置 spec.tikvGCLifeTime 和 spec.from 字段，并且可以省略以下创建 backup-demo1-tidb-secret secret 的步骤和数据库账户权限步骤。

1. 下载文件 backup-rbac.yaml，并执行以下命令在 test1 这个 namespace 中创建备份需要的 RBAC 相关资源：

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

2. 远程存储访问授权。

   如果使用 Amazon S3 来备份集群，可以使用三种权限授予方式授予权限，参考AWS 账号授权授权访问兼容 S3 的远程存储；使用 Ceph 作为后端存储测试备份时，是通过 AccessKey 和 SecretKey 模式授权，设置方式可参考通过 AccessKey 和 SecretKey 授权。

3. 创建 backup-demo1-tidb-secret secret。该 secret 存放用于访问 TiDB 集群的用户所对应的密码。

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=${password} --namespace=test1
```

数据库账户权限

- mysql.tidb 表的 SELECT 和 UPDATE 权限：备份前后，Backup CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间

使用 BR 备份数据到 Amazon S3 的存储

- 创建 Backup CR，通过 accessKey 和 secretKey 授权的方式备份集群:

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

backup-aws-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  br:
    cluster: demo1
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
    # options:
    # - --lastbackupts=420134118382108673
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

- 创建 Backup CR，通过 IAM 绑定 Pod 授权的方式备份集群:

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

backup-aws-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
```

```
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  backupType: full
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # options:
    # - --lastbackupts=420134118382108673
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

- 创建 Backup CR，通过 IAM 绑定 ServiceAccount 授权的方式备份集群:

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

backup-aws-s3.yaml 文件内容如下：

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

```
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # options:
    # - --lastbackupts=420134118382108673
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

以上三个示例分别使用三种授权模式将数据导出到 Amazon S3 存储上。Amazon S3 的 acl、endpoint、storageClass 配置项均可以省略。兼容 S3 的存储相关配置参考S3 存储字段介绍。

以上示例中，.spec.br 中的一些参数项均可省略，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释参考BR 字段介绍。

自 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。

更多 Backup CR 字段的详细解释参考Backup CR 字段介绍。

创建好 Backup CR 后，可通过如下命令查看备份状态：

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

备份示例

备份全部集群数据

```
---
apiVersion: pingcap.com/v1alpha1
```

```
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  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: 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
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
```

```
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  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: 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
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  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: 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
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

6.6.3.1.2 定时全量备份

用户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。定时全量备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次全量备份，定时全量备份底层通过 Ad-hoc 全量备份来实现。下面是创建定时全量备份的具体步骤：

定时全量备份环境准备

同Ad-hoc 备份环境准备。

使用 BR 定时备份数据到 Amazon S3 的存储

- 创建 BackupSchedule CR，开启 TiDB 集群定时全量备份，通过 accessKey 和 secretKey 授权的方式备份集群：

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

backup-scheduler-aws-s3.yaml 文件内容如下：

```
---
apiVersion: 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
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
      # sendCredToTikv: true
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

- **创建 BackupSchedule CR，开启 TiDB 集群定时全量备份，通过 IAM 绑定 Pod 授权的方式备份集群：**

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

backup-scheduler-aws-s3.yaml 文件内容如下：

```
---
apiVersion: 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
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

- **创建 BackupSchedule CR，开启 TiDB 集群定时全量备份，通过 IAM 绑定 ServiceAccount 授权的方式备份集群：**

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

backup-scheduler-aws-s3.yaml 文件内容如下：

```
---
```

```
apiVersion: 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
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

定时全量备份创建完成后，可以通过以下命令查看定时全量备份的状态：

```
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
```

从以上示例可知，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。backupTemplate 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。backupSchedule 独有的配置项具体介绍可参考BackupSchedule CR 字段介绍。

6.6.3.1.3 删除备份的 Backup CR

删除备份的 Backup CR 可参考删除备份的 Backup CR。

6.6.3.1.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.3.2 使用 BR 恢复 S3 兼容存储上的备份数据

本文详细描述了如何将存储在 Amazon S3 存储的备份数据恢复到 AWS Kubernetes 环境中的 TiDB 集群，底层通过使用 BR 进行数据恢复。

本文使用的恢复方式基于 TiDB Operator 新版（v1.1 及以上）的 Custom Resource Definition (CRD) 实现。

以下示例将 Amazon S3 的存储（指定路径）上的备份数据恢复到 AWS Kubernetes 环境中的 TiDB 集群。

6.6.3.2.1 环境准备

> **注意：**
>
> 如果使用 TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, 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，并且可以省略以下创建 restore-demo2-tidb-secret secret 的步骤和数据库账户权限步骤。

1. 下载文件 backup-rbac.yaml，并执行以下命令在 test2 这个 namespace 中创建备份需要的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. 远程存储访问授权。

   如果从 Amazon S3 恢复集群数据，可以使用三种权限授予方式授予权限，参考AWS 账号授权授权访问兼容 S3 的远程存储；使用 Ceph 作为后端存储测试恢复时，是通过 AccessKey 和 SecretKey 模式授权，设置方式可参考通过 AccessKey 和 SecretKey 授权。

3. 创建 `restore-demo2-tidb-secret` secret。该 secret 存放用于访问 TiDB 集群的 root 账号和密钥。

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ password=${password} --namespace=test2
```

6.6.3.2.2 数据库账户权限

- `mysql.tidb` 表的 `SELECT` 和 `UPDATE` 权限：恢复前后，Restore CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间

6.6.3.2.3 将指定备份数据恢复到 TiDB 集群

- 创建 `Restore` CR，通过 accessKey 和 secretKey 授权的方式恢复集群：

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

`restore-aws-s3.yaml` 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test2
spec:
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  # to:
  #   host: ${tidb_host}
```

```
  #   port: ${tidb_port}
  #   user: ${tidb_user}
  #   secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

- 创建 Restore CR，通过 IAM 绑定 Pod 授权的方式备份集群：

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

restore-aws-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test2
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  br:
    cluster: demo2
    sendCredToTikv: false
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

- 创建 Restore CR，通过 IAM 绑定 ServiceAccount 授权的方式备份集群：

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

restore-aws-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test2
spec:
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo2
    sendCredToTikv: false
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

创建好 Restore CR 后，可通过以下命令查看恢复的状态：

```
kubectl get rt -n test2 -o wide
```

以上示例将存储在 Amazon S3 上指定路径 spec.s3.bucket 存储桶中 spec.s3.prefix 文件夹下的备份数据恢复到 namespace test2 中的 TiDB 集群 demo2。兼容 S3 的存储相关配置参考S3 存储字段介绍。

以上示例中，.spec.br 中的一些参数项均可省略，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释参考BR 字段介绍。

更多 restore CR 字段的详细解释参考Restore CR 字段介绍。

6.6.3.2.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.3.3 使用 Dumpling 备份 TiDB 集群数据到兼容 S3 的存储

本文详细描述了如何将 Kubernetes 上的 TiDB 集群数据备份到兼容 S3 的存储上。本文档中的 “备份”，均是指全量备份（Ad-hoc 全量备份和定时全量备份）。底层通过使用 Dumpling 获取集群的逻辑备份，然后在将备份数据上传到兼容 S3 的存储上。

本文使用的备份方式基于 TiDB Operator 新版（v1.1 及以上）的 CustomResourceDefinition (CRD) 实现。

6.6.3.3.1 Ad-hoc 全量备份

Ad-hoc 全量备份通过创建一个自定义的 Backup custom resource (CR) 对象来描述一次备份。TiDB Operator 根据这个 Backup 对象来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。

目前兼容 S3 的存储中，Ceph 和 Amazon S3 经测试可正常工作。下文提供了如何将 TiDB 集群的数据备份到 Ceph 和 Amazon S3 这两种存储的示例。示例假设对部署在 Kubernetes tidb-cluster 这个 namespace 中的 TiDB 集群 demo1 进行数据备份，以下是具体的操作过程。

Ad-hoc 全量备份环境准备

1. 执行以下命令，根据 backup-rbac.yaml 在 tidb-cluster 命名空间创建基于角色的访问控制 (RBAC) 资源。

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/manifests/backup/backup-rbac.yaml -n tidb-
    ↪ cluster
```

2. 远程存储访问授权。

   如果使用 Amazon S3 来备份集群，可以使用三种权限授予方式授予权限，参考AWS 账号授权授权访问兼容 S3 的远程存储；使用 Ceph 作为后端存储测试备份时，是通过 AccessKey 和 SecretKey 模式授权，设置方式可参考通过 AccessKey 和 SecretKey 授权。

3. 创建 backup-demo1-tidb-secret secret。该 secret 存放用于访问 TiDB 集群的 root 账号和密钥。

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=${password} --namespace=tidb-cluster
```

数据库账户权限

- mysql.tidb 表的 SELECT 和 UPDATE 权限：备份前后，Backup CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间。
- 全局权限：SELECT、RELOAD、LOCK TABLES、和 REPLICATION CLIENT。

以下是如何创建一个备份用户的示例:

```
CREATE USER 'backup'@'%' IDENTIFIED BY '...';
GRANT
  SELECT, RELOAD, LOCK TABLES, REPLICATION CLIENT
  ON *.*
  TO 'backup'@'%';
GRANT
  UPDATE, SELECT
  ON mysql.tidb
  TO 'backup'@'%';
```

备份数据到兼容 S3 的存储

注意：

由于 rclone 存在问题，如果使用 Amazon S3 存储备份，并且 Amazon S3 开启了 AWS-KMS 加密，需要在本节示例中的 yaml 文件里添加如下 spec.s3 ↪ .options 配置以保证备份成功：

```
spec:
  ...
  s3:
    ...
    options:
    - --ignore-checksum
```

注意：

如下所示，本节提供了存储访问的多种方法。只需使用符合你情况的方法即可。

- 通过导入 AccessKey 和 SecretKey 备份到 Amazon S3 的方法
- 通过导入 AccessKey 和 SecretKey 备份到 Ceph 的方法
- 通过绑定 IAM 与 Pod 的方式备份到 Amazon S3 的方法
- 通过绑定 IAM 与 ServiceAccount 的方式备份到 Amazon S3 的方法

- 创建 Backup CR，通过 AccessKey 和 SecretKey 授权的方式将数据备份到 Amazon S3：

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

backup-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: tidb-cluster
spec:
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    secretName: s3-secret
    region: ${region}
    bucket: ${bucket}
    # prefix: ${prefix}
    # storageClass: STANDARD_IA
    # acl: private
    # endpoint:
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

- 创建 Backup CR，通过 AccessKey 和 SecretKey 授权的方式将数据备份到 Ceph：

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

backup-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
```

```
  name: demo1-backup-s3
  namespace: tidb-cluster
spec:
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: ceph
    secretName: s3-secret
    endpoint: ${endpoint}
    # prefix: ${prefix}
    bucket: ${bucket}
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

- 创建 Backup CR，通过 IAM 绑定 Pod 授权的方式将数据备份到 Amazon S3：

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

backup-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: tidb-cluster
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  backupType: full
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
```

```
    region: ${region}
    bucket: ${bucket}
    # prefix: ${prefix}
    # storageClass: STANDARD_IA
    # acl: private
    # endpoint:
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

- 创建 Backup CR，通过 IAM 绑定 ServiceAccount 授权的方式将数据备份到 Amazon S3：

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

backup-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: tidb-cluster
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: ${region}
    bucket: ${bucket}
    # prefix: ${prefix}
    # storageClass: STANDARD_IA
    # acl: private
    # endpoint:
# dumpling:
#  options:
```

```
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

上述示例将 TiDB 集群的数据全量导出备份到 Amazon S3 和 Ceph 上。Amazon S3 的 acl、endpoint、storageClass 配置项均可以省略。其余非 Amazon S3 的但是兼容 S3 的存储均可使用和 Amazon S3 类似的配置。可参考上面例子中 Ceph 的配置，省略不需要配置的字段。更多兼容 S3 的存储相关配置参考S3 存储字段介绍。

以上示例中，.spec.dumpling 表示 Dumpling 相关的配置，可以在 options 字段指定 Dumpling 的运行参数，详情见 Dumpling 使用文档；默认情况下该字段可以不用配置。当不指定 Dumpling 的配置时，options 字段的默认值如下：

```
options:
- --threads=16
- --rows=10000
```

更多 Backup CR 字段的详细解释参考Backup CR 字段介绍。

创建好 Backup CR 后，可通过如下命令查看备份状态：

```
kubectl get bk -n tidb-cluster -owide
```

要获取一个 Backup job 的详细信息，请使用以下命令。对于此命令中的 $backup_job_name，请使用上一条命令输出中的名称。

```
kubectl describe bk -n tidb-cluster $backup_job_name
```

如果要再次运行 Ad-hoc 备份，你需要删除备份的 Backup CR 并重新创建。

6.6.3.3.2 定时全量备份

用户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。定时全量备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次全量备份，定时全量备份底层通过 Ad-hoc 全量备份来实现。下面是创建定时全量备份的具体步骤：

定时全量备份环境准备

同Ad-hoc 全量备份环境准备。

定时全量备份数据到 S3 兼容存储

注意：

由于 rclone 存在问题，如果使用 Amazon S3 存储备份，并且 Amazon S3 开启了 AWS-KMS 加密，需要在本节示例中的 yaml 文件里添加如下 spec.backupTemplate.s3.options 配置以保证备份成功：

```
spec:
  ...
  backupTemplate:
    ...
    s3:
      ...
      options:
      - --ignore-checksum
```

- 创建 BackupSchedule CR 开启 TiDB 集群的定时全量备份，通过 AccessKey 和 SecretKey 授权的方式将数据备份到 Amazon S3：

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

backup-schedule-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: tidb-cluster
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      secretName: s3-secret
      region: ${region}
      bucket: ${bucket}
```

```
    # prefix: ${prefix}
    # storageClass: STANDARD_IA
    # acl: private
    # endpoint:
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

- 创建 BackupSchedule CR 开启 TiDB 集群的定时全量备份，通过 AccessKey 和 SecretKey 授权的方式将数据备份到 Ceph：

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

backup-schedule-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-ceph
  namespace: tidb-cluster
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: ceph
      secretName: s3-secret
      endpoint: ${endpoint}
      bucket: ${bucket}
      # prefix: ${prefix}
  # dumpling:
  #  options:
  #  - --threads=16
```

```
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

- 创建 BackupSchedule CR 开启 TiDB 集群的定时全量备份，通过 IAM 绑定 Pod 授权的方式将数据备份到 Amazon S3：

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

backup-schedule-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: tidb-cluster
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: ${region}
      bucket: ${bucket}
      # prefix: ${prefix}
      # storageClass: STANDARD_IA
      # acl: private
      # endpoint:
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
```

```
    # storageClassName: local-storage
    storageSize: 10Gi
```

- 创建 BackupSchedule CR 开启 TiDB 集群的定时全量备份，通过 IAM 绑定 ServiceAccount 授权的方式将数据备份到 Amazon S3：

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

backup-schedule-s3.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: tidb-cluster
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  serviceAccount: tidb-backup-manager
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: ${region}
      bucket: ${bucket}
      # prefix: ${prefix}
      # storageClass: STANDARD_IA
      # acl: private
      # endpoint:
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

定时全量备份创建完成后，可以通过以下命令查看定时全量备份的状态：

```
kubectl get bks -n tidb-cluster -owide
```

查看定时全量备份下面所有的备份条目：

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

从以上示例可知，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。backupTemplate 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。backupSchedule 独有配置项介绍可参考BackupSchedule CR 字段介绍。

> 注意：
>
> TiDB Operator 会创建一个 PVC，这个 PVC 同时用于 Ad-hoc 全量备份和定时全量备份，备份数据会先存储到 PV，然后再上传到远端存储。如果备份完成后想要删掉这个 PVC，可以参考删除资源先把备份 Pod 删掉，然后再把 PVC 删掉。
>
> 假如备份并上传到远端存储成功，TiDB Operator 会自动删除本地的备份文件。如果上传失败，则本地备份文件将被保留。

6.6.3.3.3 删除备份的 Backup CR

删除备份的 Backup CR 可参考删除备份的 Backup CR。

6.6.3.3.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.3.4 使用 TiDB Lightning 恢复 S3 兼容存储上的备份数据

本文描述了将 Kubernetes 上通过 TiDB Operator 备份的数据恢复到 TiDB 集群的操作过程。

本文使用的恢复方式基于 TiDB Operator v1.1 及以上的 CustomResourceDefinition (CRD) 实现，底层通过使用 TiDB Lightning TiDB-backend 来恢复数据。

目前，TiDB Lightning 支持三种后端：Importer-backend、Local-backend 、TiDB- ↪ backend。关于这三种后端的区别和选择，请参阅 TiDB Lightning 文档。如果要使用 Importer-backend 或者 Local-backend 导入数据，请参考使用 TiDB Lightning 导入集群数据。

以下示例将兼容 S3 的存储（指定路径）上的备份数据恢复到 TiDB 集群。

6.6.3.4.1 环境准备

1. 下载文件 backup-rbac.yaml，并执行以下命令在 test2 这个 namespace 中创建恢复所需的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. 远程存储访问授权。

   如果从 Amazon S3 恢复集群数据，可以使用三种权限授予方式授予权限，参考AWS 账号授权授权访问兼容 S3 的远程存储；使用 Ceph 作为后端存储测试恢复时，是通过 AccessKey 和 SecretKey 模式授权，设置方式可参考通过 AccessKey 和 SecretKey 授权。

3. 创建 restore-demo2-tidb-secret secret，该 secret 存放用来访问 TiDB 集群的 root 账号和密钥：

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ user=root --from-literal=password=${password} --namespace=test2
```

6.6.3.4.2 数据库账户权限

权限	作用域
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables

6.6.3.4.3 将指定备份数据恢复到 TiDB 集群

注意：

由于 rclone 存在问题，如果使用 Amazon S3 存储备份，并且 Amazon S3 开启了 AWS-KMS 加密，需要在本节示例中的 yaml 文件里添加如下 spec.s3 ↪ .options 配置以保证备份恢复成功：

```
spec:
  ...
  s3:
    ...
```

```
  options:
  - --ignore-checksum
```

1. 创建 Restore customer resource (CR)，将制定备份数据恢复至 TiDB 集群

   - 创建 Restore custom resource (CR)，通过 AccessKey 和 SecretKey 授权的方式将指定的备份数据由 Ceph 恢复至 TiDB 集群：

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

   restore.yaml 文件内容如下：

   ```
   ---
   apiVersion: pingcap.com/v1alpha1
   kind: Restore
   metadata:
     name: demo2-restore
     namespace: test2
   spec:
     backupType: full
     to:
       host: ${tidb_host}
       port: ${tidb_port}
       user: ${tidb_user}
       secretName: restore-demo2-tidb-secret
     s3:
       provider: ceph
       endpoint: ${endpoint}
       secretName: s3-secret
       path: s3://${backup_path}
     # storageClassName: local-storage
     storageSize: 1Gi
   ```

   - 创建 Restore custom resource (CR)，通过 AccessKey 和 SecretKey 授权的方式将指定的备份数据由 Amazon S3 恢复至 TiDB 集群：

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

   restore.yaml 文件内容如下：

   ```
   ---
   apiVersion: pingcap.com/v1alpha1
   kind: Restore
   metadata:
   ```

```
  name: demo2-restore
  namespace: test2
spec:
  backupType: full
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: ${region}
    secretName: s3-secret
    path: s3://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

- 创建 Restore custom resource (CR)，通过 IAM 绑定 Pod 授权的方式将指定的备份数据恢复至 TiDB 集群：

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

restore.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  backupType: full
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: ${region}
    path: s3://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

- 创建 Restore custom resource (CR)，通过 IAM 绑定 ServiceAccount 授权的方式将指定的备份数据恢复至 TiDB 集群：

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

restore.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: ${region}
    path: s3://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

2. 创建好 Restore CR 后，可通过以下命令查看恢复的状态：

```
kubectl get rt -n test2 -owide
```

以上示例将兼容 S3 的存储（spec.s3.path 路径下）中的备份数据恢复到 TiDB 集群 spec.to.host。有关兼容 S3 的存储的配置项，可以参考S3 字段介绍。

更多 Restore CR 字段的详细解释参考Restore CR 字段介绍。

注意：

TiDB Operator 会创建一个 PVC，用于数据恢复，备份数据会先从远端存储下载到 PV，然后再进行恢复。如果恢复完成后想要删掉这个 PVC，可以参考删除资源先把恢复 Pod 删掉，然后再把 PVC 删掉。

6.6.3.4.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.4 使用 GCS 备份与恢复

6.6.4.1 使用 BR 备份 TiDB 集群到 GCS

本文档详细描述了如何将 Kubernetes 上 TiDB 集群的数据备份到 Google Cloud Storage (GCS) 上。BR 会在底层获取集群的逻辑备份，然后再将备份数据上传到远端 GCS。

本文使用的备份方式基于 TiDB Operator 新版 (v1.1 及以上) 的 CustomResourceDefinition (CRD) 实现。

6.6.4.1.1 Ad-hoc 备份

Ad-hoc 备份支持全量备份与增量备份。Ad-hoc 备份通过创建一个自定义的 Backup custom resource (CR) 对象来描述一次备份。TiDB Operator 根据这个 Backup 对象来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。

为了更好地描述备份的使用方式，本文档提供如下备份示例。示例假设对部署在 Kubernetes test1 这个 namespace 中的 TiDB 集群 demo1 进行数据备份，下面是具体操作过程。

Ad-hoc 备份环境准备

> **注意：**
>
> 如果使用 TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR 会自动调整 tikv_gc_life_time 参数，不需要在 Backup CR 中配置 spec.tikvGCLifeTime 和 spec.from 字段，并且可以省略以下创建 backup-demo1-tidb-secret secret 的步骤和数据库账户权限步骤。

1. 下载文件 backup-rbac.yaml，并执行以下命令在 test1 这个 namespace 中创建备份需要的 RBAC 相关资源：

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

2. 远程存储访问授权。

   参考GCS 账号授权授权访问 GCS 远程存储。

3. 创建 backup-demo1-tidb-secret secret。该 secret 存放用于访问 TiDB 集群的 root 账号和密钥。

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=<password> --namespace=test1
```

数据库账户权限

- mysql.tidb 表的 SELECT 和 UPDATE 权限：备份前后，Backup CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间

Ad-hoc 备份过程

1. 创建 Backup CR，并将数据备份到 GCS：

```
kubectl apply -f backup-gcs.yaml
```

backup-gcs.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
    # sendCredToTikv: true
    # options:
    # - --lastbackupts=420134118382108673
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

以上示例中，spec.br 中的一些参数项均可省略，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释参考BR 字段介绍。

自 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。

该示例将 TiDB 集群的数据全量导出备份到 GCS。spec.gcs 中的一些参数项均可省略，如 location、objectAcl、storageClass。GCS 存储相关配置参考GCS 存储字段介绍。

更多 Backup CR 字段的详细解释参考Backup CR 字段介绍。

2. 创建好 Backup CR 后，可通过以下命令查看备份状态：

```
kubectl get bk -n test1 -owide
```

备份示例

备份全部集群数据

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
  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: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    clusterNamespace: test1
  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: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
```

```
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    clusterNamespace: test1
  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: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  br:
    cluster: demo1
    clusterNamespace: test1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
```

```
  # location: us-east1
  # storageClass: STANDARD_IA
  # objectAcl: private
```

6.6.4.1.2 定时全量备份

用户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。定时全量备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次全量备份，定时全量备份底层通过 Ad-hoc 全量备份来实现。下面是创建定时全量备份的具体步骤：

定时全量备份环境准备

同Ad-hoc 全量备份环境准备。

定时全量备份过程

1. 创建 BackupSchedule CR，开启 TiDB 集群的定时全量备份，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: 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
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    br:
      cluster: demo1
      clusterNamespace: test1
```

```
    # 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
```

2. 定时全量备份创建完成后，通过以下命令查看备份的状态：

```
kubectl get bks -n test1 -owide
```

查看定时全量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
```

从以上示例可知，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。backupTemplate 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。backupSchedule 独有的配置项具体介绍可参考BackupSchedule CR 字段介绍。

6.6.4.1.3 删除备份的 Backup CR

删除备份的 Backup CR 可参考删除备份的 Backup CR。

6.6.4.1.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.4.2 使用 BR 恢复 GCS 上的备份数据

本文描述了如何将存储在 Google Cloud Storage (GCS) 上的备份数据恢复到 Kubernetes 环境中的 TiDB 集群。底层通过使用 BR 来进行集群恢复。

本文使用的恢复方式基于 TiDB Operator 新版 (v1.1 及以上) 的 CustomResourceDefinition (CRD) 实现。

以下示例将存储在 GCS 上指定路径的集群备份数据恢复到 TiDB 集群。

6.6.4.2.1 环境准备

> **注意：**
>
> 如果使用 TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, 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，并且可以省略以下创建 `restore-demo2-tidb-secret` secret 的步骤和数据库账户权限步骤。

1. 下载文件 `backup-rbac.yaml`，并执行以下命令在 `test2` 这个 namespace 中创建恢复所需的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. 远程存储访问授权。

   参考GCS 账号授权授权访问 GCS 远程存储。

3. 创建 `restore-demo2-tidb-secret` secret，该 secret 存放用来访问 TiDB 集群的 root 账号和密钥：

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ user=root --from-literal=password=<password> --namespace=test2
```

6.6.4.2.2 数据库账户权限

- `mysql.tidb` 表的 `SELECT` 和 `UPDATE` 权限：恢复前后，Restore CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间

6.6.4.2.3 恢复过程

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.yaml
```

   `restore.yaml` 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-gcs
  namespace: test2
```

```
spec:
  # backupType: full
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
    # sendCredToTikv: true
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  # to:
  #   host: ${tidb_host}
  #   port: ${tidb_port}
  #   user: ${tidb_user}
  #   secretName: restore-demo2-tidb-secret
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

2. 创建好 Restore CR 后，通过以下命令查看恢复的状态：

```
shell kubectl get rt -n test2 -owide
```

以上示例将存储在 GCS 上指定路径 spec.gcs.bucket 存储桶中 spec.gcs.prefix 文件夹下的备份数据恢复到 namespace test2 中的 TiDB 集群 demo2。GCS 存储相关配置参考GCS 存储字段介绍。

以上示例中，.spec.br 中的一些参数项均可省略，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释参考BR 字段介绍。

更多 Restore CR 字段的详细解释参考Restore CR 字段介绍。

6.6.4.2.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.4.3 使用 Dumpling 备份 TiDB 集群数据到 GCS

本文档详细描述了如何将 Kubernetes 上 TiDB 集群的数据备份到 Google Cloud Storage (GCS) 上。本文档中的“备份”，均是指全量备份（Ad-hoc 全量备份和定时全量备份），底层通过使用 Dumpling 获取集群的逻辑备份，然后再将备份数据上传到远端 GCS。

本文使用的备份方式基于 TiDB Operator 新版（v1.1 及以上）的 CustomResourceDefinition (CRD) 实现。

6.6.4.3.1 数据库账户权限

- mysql.tidb 表的 SELECT 和 UPDATE 权限：备份前后，Backup CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间
- SELECT
- RELOAD
- LOCK TABLES
- REPLICATION CLIENT

6.6.4.3.2 Ad-hoc 全量备份

Ad-hoc 全量备份通过创建一个自定义的 Backup custom resource (CR) 对象来描述一次备份。TiDB Operator 根据这个 Backup 对象来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。

为了更好地描述备份的使用方式，本文档提供如下备份示例。示例假设对部署在 Kubernetes test1 这个 namespace 中的 TiDB 集群 demo1 进行数据备份，下面是具体操作过程。

Ad-hoc 全量备份环境准备

1. 下载文件 backup-rbac.yaml，并执行以下命令在 test1 这个 namespace 中创建备份需要的 RBAC 相关资源：

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

2. 远程存储访问授权。

   参考GCS 账号授权授权访问 GCS 远程存储。

3. 创建 backup-demo1-tidb-secret secret。该 secret 存放用于访问 TiDB 集群的 root 账号和密钥。

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=${password} --namespace=test1
```

备份数据到 GCS

1. 创建 Backup CR，并将数据备份到 GCS：

```
kubectl apply -f backup-gcs.yaml
```

backup-gcs.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  gcs:
    secretName: gcs-secret
    projectId: ${project_id}
    bucket: ${bucket}
    # prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
    # bucketAcl: private
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  storageClassName: local-storage
  storageSize: 10Gi
```

以上示例将 TiDB 集群的数据全量导出备份到 GCS。GCS 配置中的 location、objectAcl、bucketAcl、storageClass 项均可以省略。GCS 存储相关配置参考GCS 存储字段介绍。

以上示例中的 .spec.dumpling 表示 Dumpling 相关的配置，可以在 options 字段指定 Dumpling 的运行参数，详情见 Dumpling 使用文档；默认情况下该字段可以不用配置。当不指定 Dumpling 的配置时，options 字段的默认值如下：

```
options:
- --threads=16
- --rows=10000
```

更多 Backup CR 字段的详细解释参考Backup CR 字段介绍。

2. 创建好 Backup CR 后，可通过以下命令查看备份状态：

```
kubectl get bk -n test1 -owide
```

6.6.4.3.3 定时全量备份

用户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。定时全量备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次全量备份，定时全量备份底层通过 Ad-hoc 全量备份来实现。下面是创建定时全量备份的具体步骤：

定时全量备份环境准备

同Ad-hoc 全量备份环境准备。

定时全量备份数据到 GCS

1. 创建 BackupSchedule CR 开启 TiDB 集群的定时全量备份，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: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-gcs
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    gcs:
      secretName: gcs-secret
      projectId: ${project_id}
      bucket: ${bucket}
      # prefix: ${prefix}
      # location: us-east1
```

```
    # storageClass: STANDARD_IA
    # objectAcl: private
    # bucketAcl: private
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

2. 定时全量备份创建完成后，可以通过以下命令查看定时全量备份的状态：

```
kubectl get bks -n test1 -owide
```

查看定时全量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
```

从以上示例可知，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。backupTemplate 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。backupSchedule 独有的配置项具体介绍可参考BackupSchedule CR 字段介绍。

> **注意：**
>
> TiDB Operator 会创建一个 PVC，这个 PVC 同时用于 Ad-hoc 全量备份和定时全量备份，备份数据会先存储到 PV，然后再上传到远端存储。如果备份完成后想要删掉这个 PVC，可以参考删除资源先把备份 Pod 删掉，然后再把 PVC 删掉。
>
> 假如备份并上传到远端存储成功，TiDB Operator 会自动删除本地的备份文件。如果上传失败，则本地备份文件将被保留。

6.6.4.3.4 删除备份的 Backup CR

删除备份的 Backup CR 可参考删除备份的 Backup CR。

6.6.4.3.5 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.4.4 使用 TiDB Lightning 恢复 GCS 上的备份数据

本文描述了将 Kubernetes 上通过 TiDB Operator 备份的数据恢复到 TiDB 集群的操作过程。

本文使用的恢复方式基于 TiDB Operator v1.1 及以上的 CustomResourceDefinition (CRD) 实现，底层通过使用 TiDB Lightning TiDB-backend 来恢复数据。

目前，TiDB Lightning 支持三种后端：`Importer-backend`、`Local-backend` 、`TiDB-backend`。关于这三种后端的区别和选择，请参阅 TiDB Lightning 文档。如果要使用 `Importer-backend` 或者 `Local-backend` 导入数据，请参阅使用 TiDB Lightning 导入集群数据。

以下示例将存储在 Google Cloud Storage (GCS) 上指定路径上的集群备份数据恢复到 TiDB 集群。

6.6.4.4.1 环境准备

1. 下载文件 `backup-rbac.yaml`，并执行以下命令在 `test2` 这个 namespace 中创建恢复所需的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. 远程存储访问授权。

   参考GCS 账号授权授权访问 GCS 远程存储。

3. 创建 `restore-demo2-tidb-secret` secret，该 secret 存放用来访问 TiDB 集群的 root 账号和密钥：

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ user=root --from-literal=password=${password} --namespace=test2
```

6.6.4.4.2 数据库账户权限

权限	作用域
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables

6.6.4.4.3 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.yaml
```

restore.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
spec:
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    path: gcs://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

以上示例将存储在 GCS 上指定路径 spec.gcs.path 的备份数据恢复到 TiDB 集群 spec.to.host。关于 GCS 的配置项可以参考GCS 字段介绍。

更多 Restore CR 字段的详细解释参考Restore CR 字段介绍。

2. 创建好 Restore CR 后可通过以下命令查看恢复的状态：

```
shell kubectl get rt -n test2 -owide
```

> **注意：**
>
> TiDB Operator 会创建一个 PVC，用于数据恢复，备份数据会先从远端存储下载到 PV，然后再进行恢复。如果恢复完成后想要删掉这个 PVC，可以参考删除资源先把恢复 Pod 删掉，然后再把 PVC 删掉。

6.6.4.4.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.5 使用持久卷备份与恢复

6.6.5.1 使用 BR 备份 TiDB 集群到持久卷

本文档详细描述了如何将 Kubernetes 上 TiDB 集群的数据备份到持久卷上。

本文描述的持久卷指任何 Kubernetes 支持的持久卷类型。本文将以 NFS 为例，介绍如何使用 BR 备份 TiDB 集群的数据到持久卷。

> **注意：**
>
> 本文使用的备份方式需要 TiDB Operator v1.1.8 及以上才支持。

6.6.5.1.1 Ad-hoc 备份

Ad-hoc 备份支持全量备份与增量备份。Ad-hoc 备份通过创建一个自定义的 `Backup` custom resource (CR) 对象来描述一次备份。TiDB Operator 根据这个 `Backup` 对象来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。

为了更好地描述备份的使用方式，本文档提供如下备份示例。示例假设对部署在 Kubernetes `test1` 这个 namespace 中的 TiDB 集群 `demo1` 进行数据备份，下面是具体操作过程。

Ad-hoc 备份环境准备

> **注意：**
>
> 如果使用 TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR 会自动调整 `tikv_gc_life_time` 参数，不需要在 Backup CR 中配置 `spec.tikvGCLifeTime` 和 `spec.from` 字段，并且可以省略以下创建 `backup-demo1-tidb-secret` secret 的步骤和数据库账户权限步骤。

1. 下载文件 backup-rbac.yaml，并执行以下命令在 `test1` 这个 namespace 中创建备份需要的 RBAC 相关资源：

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

2. 创建 `backup-demo1-tidb-secret` secret。该 secret 存放用于访问 TiDB 集群的账号的密码。

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=<password> --namespace=test1
```

3. 确认可以从 Kubernetes 集群中访问用于存储备份数据的 NFS 服务器，并且配置了 TiKV 挂载跟备份任务相同的 NFS 共享目录到相同的本地目录。TiKV 挂载 NFS 的具体配置方法可以参考如下配置：

```
spec:
  tikv:
    additionalVolumes:
    # specify volume types that are supported by Kubernetes, Ref: https
        ↪ ://kubernetes.io/docs/concepts/storage/persistent-volumes/#
        ↪ types-of-persistent-volumes
    - name: nfs
      nfs:
        server: 192.168.0.2
        path: /nfs
    additionalVolumeMounts:
    # this must match `name` in `additionalVolumes`
    - name: nfs
      mountPath: /nfs
```

数据库账户权限

- mysql.tidb 表的 SELECT 和 UPDATE 权限：备份前后，Backup CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间

Ad-hoc 备份过程

1. 创建 Backup CR，并将数据备份到 NFS：

```
kubectl apply -f backup-nfs.yaml
```

backup-nfs.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  # from:
  #   host: ${tidb-host}
  #   port: ${tidb-port}
  #   user: ${tidb-user}
```

```
#   secretName: backup-demo1-tidb-secret
br:
  cluster: demo1
  clusterNamespace: test1
  # logLevel: info
  # statusAddr: ${status-addr}
  # concurrency: 4
  # rateLimit: 0
  # checksum: true
  # options:
  # - --lastbackupts=420134118382108673
local:
  prefix: backup-nfs
  volume:
    name: nfs
    nfs:
      server: ${nfs_server_ip}
      path: /nfs
  volumeMount:
    name: nfs
    mountPath: /nfs
```

以上示例中的 .spec.local 表示持久卷相关配置，详细解释参考Local 存储字段介绍。

以上示例中，spec.br 中的一些参数项均可省略，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo。更多 .spec.br 字段的详细解释参考BR 字段介绍。

自 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。

更多 Backup CR 字段的详细解释参考Backup CR 字段介绍。

该示例将 TiDB 集群的数据全量导出备份到 NFS。

2. 创建好 Backup CR 后，可通过以下命令查看备份状态：

```
kubectl get bk -n test1 -owide
```

备份示例

备份全部集群数据

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
```

```
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

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: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
```

```
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

备份单张表的数据

以下示例中，备份 `db1.table1` 表的数据。

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

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: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  br:
    cluster: demo1
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

6.6.5.1.2 定时全量备份

用户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。定时全量备份通过自定义的 BackupSchedule CR 对象来描述。每到备份时间点会触发一次全量备份，定时全量备份底层通过 Ad-hoc 全量备份来实现。下面是创建定时全量备份的具体步骤：

定时全量备份环境准备

同Ad-hoc 全量备份环境准备。

定时全量备份过程

1. 创建 BackupSchedule CR，开启 TiDB 集群的定时全量备份，将数据备份到 NFS：

```
kubectl apply -f backup-schedule-nfs.yaml
```

backup-schedule-nfs.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-nfs
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    # from:
    #   host: ${tidb_host}
    #   port: ${tidb_port}
    #   user: ${tidb_user}
    #   secretName: backup-demo1-tidb-secret
    br:
      cluster: demo1
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status-addr}
      # concurrency: 4
      # rateLimit: 0
      # checksum: true
    local:
      prefix: backup-nfs
      volume:
        name: nfs
        nfs:
          server: ${nfs_server_ip}
          path: /nfs
      volumeMount:
        name: nfs
        mountPath: /nfs
```

2. 定时全量备份创建完成后，通过以下命令查看备份的状态：

```
kubectl get bks -n test1 -owide
```

查看定时全量备份下面所有的备份条目：

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

从以上示例可知，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。backupTemplate 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。backupSchedule 独有的配置项具体介绍可参考BackupSchedule CR 字段介绍。

6.6.5.1.3 删除备份的 Backup CR

删除备份的 Backup CR 可参考删除备份的 Backup CR。

6.6.5.1.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.6.5.2 使用 BR 恢复持久卷上的备份数据

本文描述了如何将存储在持久卷上的备份数据恢复到 Kubernetes 环境中的 TiDB 集群。底层通过使用 BR 来进行集群恢复。

本文描述的持久卷指任何 Kubernetes 支持的持久卷类型。以下示例以 NFS 存储卷为例，介绍如何将存储在持久卷上指定路径的集群备份数据恢复到 TiDB 集群。

本文使用的恢复方式基于 TiDB Operator 新版 (v1.1.8 及以上) 的 CustomResourceDefinition (CRD) 实现。

6.6.5.2.1 环境准备

> **注意：**
>
> 如果使用 TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, 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，并且可以省略以下创建 restore-demo2-tidb-secret secret 的步骤和数据库账户权限步骤。

1. 下载文件 backup-rbac.yaml，并执行以下命令在 test2 这个 namespace 中创建恢复所需的 RBAC 相关资源：

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. 创建 restore-demo2-tidb-secret secret，该 secret 存放用来访问 TiDB 服务的账号的密码：

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ user=root --from-literal=password=<password> --namespace=test2
```

3. 确认可以从 Kubernetes 集群中访问用于存储备份数据的 NFS 服务器。

6.6.5.2.2 数据库账户权限

- mysql.tidb 表的 SELECT 和 UPDATE 权限：恢复前后，Restore CR 需要一个拥有该权限的数据库账户，用于调整 GC 时间

6.6.5.2.3 恢复过程

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.yaml
```

restore.yaml 文件内容如下：

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-nfs
  namespace: test2
spec:
  # backupType: full
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  # to:
  #   host: ${tidb_host}
  #   port: ${tidb_port}
  #   user: ${tidb_user}
  #   secretName: restore-demo2-tidb-secret
  local:
    prefix: backup-nfs
```

```
  volume:
    name: nfs
    nfs:
      server: ${nfs_server_if}
      path: /nfs
  volumeMount:
    name: nfs
    mountPath: /nfs
```

2. 创建好 Restore CR 后，通过以下命令查看恢复的状态：

```
kubectl get rt -n test2 -owide
```

以上示例将存储在 NFS 上指定路径 local://${.spec.local.volumeMount.mountPath}/${.spec.local.prefix}/ 文件夹下的备份数据恢复到 namespace test2 中的 TiDB 集群 demo2。持久卷存储相关配置参考Local 存储字段介绍。

以上示例中，.spec.br 中的一些参数项均可省略，如 logLevel、statusAddr、concurrency、rateLimit、checksum、timeAgo、sendCredToTikv。更多 .spec.br 字段的详细解释参考BR 字段介绍。

更多 Restore CR 字段的详细解释参考Restore CR 字段介绍。

6.6.5.2.4 故障诊断

在使用过程中如果遇到问题，可以参考故障诊断。

6.7 重启 Kubernetes 上的 TiDB 集群

在使用 TiDB 集群的过程中，如果你发现某个 Pod 存在内存泄漏等问题，需要对集群进行重启，本文描述了如何优雅滚动重启 TiDB 集群内某个组件的所有 Pod 或通过优雅重启指令来将 TiDB 集群内某个 Pod 优雅下线然后再进行重新启动。

> **警告：**
>
> 在生产环境中，未经过优雅重启而手动删除某个 TiDB 集群 Pod 节点是一件极其危险的事情，虽然 StatefulSet 控制器会将 Pod 节点再次拉起，但这依旧可能会引起部分访问 TiDB 集群的请求失败。

6.7.1 优雅滚动重启 TiDB 集群组件的所有 Pod 节点

在标准 Kubernetes 上部署 TiDB 集群之后，通过 `kubectl edit tc ${name} -n ${namespace}` 修改集群配置，为期望优雅滚动重启的 TiDB 集群组件 Spec 添加 annotation `tidb.pingcap.com/restartedAt`，Value 设置为当前时间。以下示例中，为组件 `pd`、`tikv`、`tidb` 都设置了 annotation，表示将优雅滚动重启以上三个 TiDB 集群组件的所有 Pod。可以根据实际情况，只为某个组件设置 annotation。

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
  tikv:
    baseImage: pingcap/tikv
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
    config: {}
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
```

6.8 维护 TiDB 集群所在的 Kubernetes 节点

TiDB 是高可用数据库，可以在部分数据库节点下线的情况下正常运行，因此，我们可以安全地对底层 Kubernetes 节点进行停机维护。在具体操作时，针对 PD、TiKV 和 TiDB Pod 的不同特性，我们需要采取不同的操作策略。

本文档将详细介绍如何对 Kubernetes 节点进行临时或长期的维护操作。

环境准备：

- kubectl
- jq

> 注意：
> 维护节点前，需要保证 Kubernetes 集群的剩余资源足够运行 TiDB 集群。

6.8.1 维护短期内可恢复的节点

1. 使用 `kubectl cordon` 命令标记待维护节点为不可调度，防止新的 Pod 调度到待维护节点上：

```
kubectl cordon ${node_name}
```

2. 检查待维护节点上是否有 TiKV Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

假如存在 TiKV Pod，针对每一个 Pod，进行以下操作：

1. 参考迁移 TiKV Region Leader 将 Region Leader 迁移到其他 Pod。
2. 通过调整 PD 的 `max-store-down-time` 配置来增大集群所允许的 TiKV Pod 下线时间，在此时间内维护完毕并恢复 Kubernetes 节点后，所有该节点上的 TiKV Pod 会自动恢复。

   以调整 `max-store-down-time` 为 `60m` 为例，请使用以下命令。你可以调整 `max-store-down-time` 到合理的值。

```
pd-ctl config set max-store-down-time 60m
```

3. 检查待维护节点上是否有 PD Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

假如存在 PD Pod，针对每一个 Pod，参考迁移 PD Leader 将 Leader 迁移到其他 Pod。

4. 确认待维护节点上不再有 TiKV 和 PD Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

5. 使用 kubectl drain 命令将待维护节点上的 Pod 迁移到其它节点上：

```
kubectl drain ${node_name} --ignore-daemonsets
```

运行后，该节点上的 Pod 会自动迁移到其它可用节点上。

6. 再次确认节点不再有任何 TiKV、TiDB 和 PD Pod 运行：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

7. 如果节点维护结束，在恢复节点后确认其健康状态：

```
watch kubectl get node ${node_name}
```

观察到节点进入 Ready 状态后，继续操作。

8. 使用 kubectl uncordon 命令解除节点的调度限制：

```
kubectl uncordon ${node_name}
```

9. 观察 Pod 是否全部恢复正常运行：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

Pod 恢复正常运行后，操作结束。

6.8.2 维护短期内不可恢复的节点

1. 检查待维护节点上是否有 TiKV Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

假如存在 TiKV Pod，针对每一个 Pod，参考重调度 TiKV Pod 将 Pod 重调度到其他节点。

2. 检查待维护节点上是否有 PD Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

假如存在 PD Pod，针对每一个 Pod，参考重调度 PD Pod 将 Pod 重调度到其他节点。

3. 确认待维护节点上不再有 TiKV 和 PD Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

4. 使用 `kubectl drain` 命令将待维护节点上的 Pod 迁移到其它节点上：

```
kubectl drain ${node_name} --ignore-daemonsets
```

运行后，该节点上的 Pod 会自动迁移到其它可用节点上。

5. 再次确认节点不再有任何 TiKV、TiDB 和 PD Pod 运行：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

6. 最后（可选），假如是长期下线节点，建议将节点从 Kubernetes 集群中删除：

```
kubectl delete node ${node_name}
```

6.8.3 重调度 PD Pod

针对节点长期下线等情形，为了尽可能减少业务受到的影响，可以将该节点上的 PD Pod 预先调度到其他节点。

6.8.3.1 如果节点存储可自动迁移

如果节点存储可以自动迁移（比如使用 EBS），你不需要删除 PD Member，只需要迁移 Leader 到其他 Pod 后删除原来的 Pod 就可以实现重调度。

1. 使用 `kubectl cordon` 命令标记待维护节点为不可调度，防止新的 Pod 调度到待维护节点上：

```
kubectl cordon ${node_name}
```

2. 查看待维护节点上的 PD Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

3. 参考迁移 PD Leader 将 Leader 迁移到其他 Pod。

4. 删除 PD Pod：

```
kubectl delete -n ${namespace} pod ${pod_name}
```

5. 确认该 PD Pod 正常调度到其它节点上：

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

6.8.3.2 如果节点存储不可自动迁移

如果节点存储不可以自动迁移（比如使用本地存储），你需要删除 PD Member 以实现重调度。

1. 使用 `kubectl cordon` 命令标记待维护节点为不可调度，防止新的 Pod 调度到待维护节点上：

```
kubectl cordon ${node_name}
```

2. 查看待维护节点上的 PD Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

3. 参考迁移 PD Leader 将 Leader 迁移到其他 Pod。

4. 下线 PD Pod。

```
pd-ctl member delete name ${pod_name}
```

5. 确认 PD Member 已删除：

```
pd-ctl member
```

6. 解除 PD Pod 与节点本地盘的绑定。

   查询 Pod 使用的 `PersistentVolumeClaim`：

```
kubectl -n ${namespace} get pvc -l tidb.pingcap.com/pod-name=${pod_name
    ↪ }
```

   删除该 `PersistentVolumeClaim`：

```
kubectl delete -n ${namespace} pvc ${pvc_name} --wait=false
```

7. 删除 PD Pod：

```
kubectl delete -n ${namespace} pod ${pod_name}
```

8. 观察该 PD Pod 是否正常调度到其它节点上：

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

6.8.4 重调度 TiKV Pod

针对节点长期下线等情形，为了尽可能减少业务受到的影响，可以将该节点上的 TiKV Pod 预先调度到其他节点。

6.8.4.1 如果节点存储可自动迁移

如果节点存储可以自动迁移（比如使用 EBS），你不需要删除整个 TiKV Store，只需要迁移 Region Leader 到其他 Pod 后删除原来的 Pod 就可以实现重调度。

1. 使用 `kubectl cordon` 命令标记待维护节点为不可调度，防止新的 Pod 调度到待维护节点上：

```
kubectl cordon ${node_name}
```

2. 查看待维护节点上的 TiKV Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

3. 参考迁移 TiKV Region Leader 将 Leader 迁移到其他 Pod。

4. 删除 TiKV Pod：

```
kubectl delete -n ${namespace} pod ${pod_name}
```

5. 确认该 TiKV Pod 正常调度到其它节点上：

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

6. 移除 evict-leader-scheduler，等待 Region Leader 自动调度回来：

```
pd-ctl scheduler remove evict-leader-scheduler-${ID}
```

6.8.4.2 如果节点存储不可自动迁移

如果节点存储不可以自动迁移（比如使用本地存储），你需要删除整个 TiKV Store 以实现重调度。

1. 使用 `kubectl cordon` 命令标记待维护节点为不可调度，防止新的 Pod 调度到待维护节点上：

```
kubectl cordon ${node_name}
```

2. 查看待维护节点上的 TiKV Pod：

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

3. 参考迁移 TiKV Region Leader 将 Leader 迁移到其他 Pod。

4. 下线 TiKV Pod。

> **注意：**
>
> 下线 TiKV Pod 前，需要保证集群中剩余的 TiKV Pod 数不少于 PD 配置中的 TiKV 数据副本数（配置项：`max-replicas`，默认值 3）。假如不符合该条件，需要先操作扩容 TiKV。

查看 TiKV Pod 的 `store-id`：

```
kubectl get tc ${cluster_name} -ojson | jq ".status.tikv.stores | .[] |
    ↪  select ( .podName == \"${pod_name}\" ) | .id"
```

下线 Pod：

```
pd-ctl store delete ${ID}
```

5. 等待 store 状态（`state_name`）转移为 `Tombstone`：

```
watch pd-ctl store ${ID}
```

6. 解除 TiKV Pod 与节点本地盘的绑定。

查询 Pod 使用的 `PersistentVolumeClaim`：

```
kubectl -n ${namespace} get pvc -l tidb.pingcap.com/pod-name=${pod_name
    ↪ }
```

删除该 `PersistentVolumeClaim`：

```
kubectl delete -n ${namespace} pvc ${pvc_name} --wait=false
```

7. 删除 TiKV Pod：

```
kubectl delete -n ${namespace} pod ${pod_name}
```

8. 观察该 TiKV Pod 是否正常调度到其它节点上：

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

9. 移除 evict-leader-scheduler，等待 Region Leader 自动调度回来：

```
pd-ctl scheduler remove evict-leader-scheduler-${ID}
```

6.8.5 迁移 PD Leader

1. 查看 PD Leader：

```
pd-ctl member leader show
```

2. 如果 Leader Pod 所在节点是要维护的节点，则需要将 Leader 先迁移到其他节点上的 Pod。

```
pd-ctl member leader transfer ${pod_name}
```

其中 `${pod_name}` 是其他节点上的 PD Pod。

6.8.6 迁移 TiKV Region Leader

1. 查看 TiKV Pod 的 `store-id`：

```
kubectl get tc ${cluster_name} -ojson | jq ".status.tikv.stores | .[] |
    ↪ select ( .podName == \"${pod_name}\" ) | .id"
```

2. 驱逐 Region Leader：

```
pd-ctl scheduler add evict-leader-scheduler ${ID}
```

3. 检查 Region Leader 已经全部被迁移走：

```
kubectl get tc ${cluster_name} -ojson | jq ".status.tikv.stores | .[] |
    ↪ select ( .podName == \"${pod_name}\" ) | .leaderCount"
```

6.9 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。

6.9.1 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。

6.9.2 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。

6.10 Kubernetes 上的 TiDB 集群故障自动转移

TiDB Operator 基于 StatefulSet 管理 Pod 的部署和扩缩容，但 `StatefulSet` 在某些 Pod 或者节点发生故障时不会自动创建新 Pod 来替换旧 Pod。为解决此问题，TiDB Operator 支持通过自动扩容 Pod 实现故障自动转移功能。

6.10.1 配置故障自动转移

故障自动转移功能在 TiDB Operator 中默认开启。

部署 TiDB Operator 时，可以在 `charts/tidb-operator/values.yaml` 文件中配置 TiDB 集群中 PD、TiKV、TiDB 和 TiFlash 组件故障转移的等待超时时间。示例如下：

```
controllerManager:
 ...
 # autoFailover is whether tidb-operator should auto failover when failure
    ↪ occurs
 autoFailover: true
 # pd failover period default(5m)
 pdFailoverPeriod: 5m
 # tikv failover period default(5m)
 tikvFailoverPeriod: 5m
 # tidb failover period default(5m)
 tidbFailoverPeriod: 5m
 # tiflash failover period default(5m)
 tiflashFailoverPeriod: 5m
```

其中，`pdFailoverPeriod`、`tikvFailoverPeriod`、`tiflashFailoverPeriod` 和 `tidbFailoverPeriod` 代表在确认实例故障后的等待超时时间，默认均为 5 分钟。超过这个时间后，TiDB Operator 就开始做故障自动转移。

另外，在配置 TiDB 集群时，可以通过 `spec.${component}.maxFailoverCount` 指定 TiDB Operator 在各组件故障自动转移时能扩容的 Pod 数量阈值，详情请参考TiDB 组件配置文档。

> **注意：**
>
> 如果集群中没有足够的资源以供 TiDB Operator 扩容新 Pod，则扩容出的 Pod 会处于 Pending 状态。

6.10.2 实现原理

TiDB 集群包括 PD、TiKV、TiDB、TiFlash、TiCDC 和 Pump 六个组件。目前 TiCDC 和 Pump 并不支持故障自动转移，PD、TiKV、TiDB 和 TiFlash 的故障转移策略有所不同，本节将详细介绍这几种策略。

6.10.2.1 PD 故障转移策略

TiDB Operator 通过 `pd/health` PD API 获取 PD members 健康状况，并记录到 TidbCluster CR 的 `.status.pd.members` 字段中。

以一个有 3 个 Pod 的 PD 集群为例，如果其中一个 Pod 不健康超过 5 分钟 (`pdFailoverPeriod` 可配置)，TiDB Operator 将自动进行以下操作：

1. TiDB Operator 将此 Pod 信息记录到 TidbCluster CR 的 `.status.pd.failureMembers` 字段中。
2. TiDB Operator 将此 Pod 下线：TiDB Operator 调用 PD API 将此 Pod 从 member 列表中删除，然后删掉 Pod 及其 PVC。
3. StatefulSet controller 会重新创建此 Pod 并以新的 member 身份加入集群。
4. 在计算 PD StatefulSet 的 Replicas 时，TiDB Operator 会将已经被删除过的 `.status.pd.failureMembers` 考虑在内，因此会扩容一个新的 Pod。此时将有 4 个 Pod 同时存在。

当原来集群中所有不健康的 Pod 都恢复正常时，TiDB Operator 会将新扩容的 Pod 自动缩容掉，恢复成原来的 Pod 数量。

> **注意：**
>
> - TiDB Operator 会为每个 PD 集群最多扩容 `spec.pd.maxFailoverCount` (默认 3) 个 Pod，超过这个阈值后不会再进行故障转移。
> - 如果 PD 集群多数 member 已经不健康，导致 PD 集群不可用，TiDB Operator 不会为这个 PD 集群进行故障自动转移。

6.10.2.2 TiDB 故障转移策略

TiDB Operator 通过访问每个 TiDB Pod 的 /status 接口确认 Pod 健康状况，并记录到 TidbCluster CR 的 .status.tidb.members 字段中。

以一个有 3 个 Pod 的 TiDB 集群为例，如果一个 Pod 不健康超过 5 分钟（tidbFailoverPeriod 可配置），TiDB Operator 将自动进行以下操作：

1. TiDB Operator 将此 Pod 信息记录到 TidbCluster CR 的 .status.tidb.failureMembers 字段中。
2. 在计算 TiDB StatefulSet 的 Replicas 时，TiDB Operator 会将 .status.tidb.failureMembers 考虑在内，因此会扩容一个新的 Pod。此时会有 4 个 Pod 同时存在。

当原来集群中不健康的 Pod 恢复正常时，TiDB Operator 会将新扩容的 Pod 缩容掉，恢复成原来的 3 个 Pod。

> **注意：**
>
> TiDB Operator 会为每个 TiDB 集群最多扩容 spec.tidb.maxFailoverCount（默认 3）个 Pod，超过这个阈值后不会再进行故障转移。

6.10.2.3 TiKV 故障转移策略

TiDB Operator 通过访问 PD API 获取 TiKV store 健康状况，并记录到 TidbCluster CR 的 .status.tikv.stores 字段中。

以一个有 3 个 Pod 的 TiKV 集群为例，当一个 TiKV Pod 无法正常工作时，该 Pod 对应的 Store 状态会变为 Disconnected。默认 30 分钟（可以通过 pd.config 中 [schedule] 部分的 max-store-down-time = "30m" 来修改）后会变成 Down 状态，然后 TiDB Operator 将自动进行以下操作：

1. 在此基础上再等待 5 分钟（可以通过 tikvFailoverPeriod 配置），如果此 TiKV Pod 仍未恢复，TiDB Operator 会将此 Pod 信息记录到 TidbCluster CR 的 .status.tikv.failureStores 字段中。
2. 在计算 TiKV StatefulSet 的 Replicas 时，TiDB Operator 会将 .status.tikv.failureStores 考虑在内，因此会扩容一个新的 Pod。此时会有 4 个 Pod 同时存在。

当原来集群中不健康的 Pod 恢复正常时，考虑到缩容 Pod 需要迁移数据，可能会对集群性能有一定影响，TiDB Operator 并不会将新扩容的 Pod 缩容掉，而是继续保持 4 个 Pod。

注意：

TiDB Operator 会为每个 TiKV 集群最多扩容 `spec.tikv.maxFailoverCount` (默认 3) 个 Pod，超过这个阈值后不会再进行故障转移。

如果所有异常的 TiKV Pod 都已经恢复，这时如果需要缩容新起的 Pod，请参考以下步骤：

配置 `spec.tikv.recoverFailover: true` (从 TiDB Operator v1.1.5 开始支持)：

```
kubectl edit tc -n ${namespace} ${cluster_name}
```

TiDB Operator 会自动将新起的 TiKV Pod 缩容，请在集群缩容完成后，配置 `spec.tikv.recoverFailover: false`，避免下次发生故障转移并恢复后自动缩容。

6.10.2.4 TiFlash 故障转移策略

TiDB Operator 通过访问 PD API 获取 TiFlash store 健康状况，并记录到 TidbCluster CR 的 `.status.tiflash.stores` 字段中。

以一个有 3 个 Pod 的 TiFlash 集群为例，当一个 TiFlash Pod 无法正常工作时，该 Pod 对应的 Store 状态会变为 `Disconnected`。默认 30 分钟 (可以通过 `pd.config` 中 `[schedule]` 部分的 `max-store-down-time = "30m"` 来修改) 后会变成 `Down` 状态，然后 TiDB Operator 将自动进行以下操作：

1. 在此基础上再等待 5 分钟 (`tiflashFailoverPeriod` 可配置)，如果此 TiFlash Pod 仍未恢复，TiDB Operator 会将此 Pod 信息记录到 TidbCluster CR 的 `.status.tiflash.failureStores` 字段中。
2. 在计算 TiFlash StatefulSet 的 Replicas 时，TiDB Operator 会将 `.status.tiflash.failureStores` 考虑在内，因此会扩容一个新的 Pod。此时会有 4 个 Pod 同时存在。

当原来集群中不健康的 Pod 恢复正常时，考虑到缩容 Pod 需要迁移数据，可能会对集群性能有一定影响，TiDB Operator 并不会将新扩容的 Pod 缩容掉，而是继续保持 4 个 Pod。

注意：

TiDB Operator 会为每个 TiFlash 集群最多扩容 `spec.tiflash.maxFailoverCount` (默认 3) 个 Pod，超过这个阈值后不会再进行故障转移。

如果所有异常的 TiFlash Pod 都已经恢复，这时如果需要缩容新起的 Pod，请参考以下步骤：

配置 `spec.tiflash.recoverFailover: true` (从 TiDB Operator v1.1.5 开始支持)：

```
kubectl edit tc -n ${namespace} ${cluster_name}
```

TiDB Operator 会自动将新起的 TiFlash Pod 缩容，请在集群缩容完成后，配置 `spec.tiflash.recoverFailover: false`，避免下次发生故障转移并恢复后自动缩容。

6.10.3 关闭故障自动转移

你可以在集群或组件级别关闭故障自动转移功能。

- 如需在集群级别关闭故障自动转移功能，在部署 TiDB Operator 时，请将 `charts/tidb-operator/values.yaml` 文件的 `controllerManager.autoFailover` 字段值配置为 `false`。示例如下：

```
controllerManager:
...
# autoFailover is whether tidb-operator should auto failover when failure occurs
autoFailover: false
```

- 如需在组件级别关闭故障自动转移功能，在创建 TiDB 集群时，请将 TidbCluster CR 中对应组件的 `spec.${component}.maxFailoverCount` 字段值配置为 `0`。

6.11 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。

6.11.1 销毁使用 TidbCluster 管理的 TiDB 集群

要销毁使用 TidbCluster 管理的 TiDB 集群，执行以下命令：

```
kubectl delete tc ${cluster_name} -n ${namespace}
```

如果集群中通过 `TidbMonitor` 部署了监控，要删除监控组件，可以执行以下命令：

```
kubectl delete tidbmonitor ${tidb_monitor_name} -n ${namespace}
```

6.11.2 销毁使用 Helm 管理的 TiDB 集群

要销毁使用 Helm 管理的 TiDB 集群，执行以下命令：

```
helm uninstall ${cluster_name} -n ${namespace}
```

6.11.3 清除数据

上述销毁集群的命令只是删除运行的 Pod，数据仍然会保留。如果你不再需要那些数据，可以通过下面命令清除数据：

> 警告：
>
> 下列命令会彻底删除数据，务必考虑清楚再执行。
>
> 为了确保数据安全，在任何情况下都不要删除 PV，除非你熟悉 PV 的工作原理。

```
kubectl delete pvc -n ${namespace} -l app.kubernetes.io/instance=${
    ↪ cluster_name},app.kubernetes.io/managed-by=tidb-operator
```

```
kubectl get pv -l app.kubernetes.io/namespace=${namespace},app.kubernetes.io
    ↪ /managed-by=tidb-operator,app.kubernetes.io/instance=${cluster_name}
    ↪ -o name | xargs -I {} kubectl patch {} -p '{"spec":{"
    ↪ persistentVolumeReclaimPolicy":"Delete"}}'
```

6.12 从 Helm 2 迁移到 Helm 3

本文以将 TiDB Operator 由 Helm 2 管理迁移到由 Helm 3 进行管理为例，介绍如何将由 Helm 2 管理的组件迁移到由 Helm 3 管理。其他如 TiDB Lightning 等由 Helm 2 管理的 release 可使用类似的步骤进行迁移。

更多有关如何将 Helm 2 管理的 release 迁移到 Helm 3 的信息，可参考 Helm 官方文档。

6.12.1 迁移步骤

假设原来由 Helm 2 管理的 TiDB Operator 安装在 `tidb-admin` namespace 下，名称为 `tidb-operator`。同时在 `tidb-cluster` namespace 下部署了名为 `basic` 的 TidbCluster 及名为 `basic` 的 TidbMonitor。

```
helm list
```

```
NAME          REVISION      UPDATED                       STATUS        CHART
    ↪                 APP VERSION   NAMESPACE
tidb-operator 1             Tue Jan 5 15:28:00 2021     DEPLOYED      tidb-
    ↪ operator-v1.1.8 v1.1.8        tidb-admin
```

1. 参考 Helm 官方文档安装 Helm 3。

   Helm 3 使用与 Helm 2 不同的配置与数据存储方式，因此在安装 Helm 3 的过程中无需担心对原有配置或数据的覆盖。

   > **注意：**
   > 安装过程中需避免 Helm 3 的 CLI binary 覆盖 Helm 2 的 CLI binary。例如，可将 Helm 3 的 CLI binary 命名为 `heml3`（本文后续示例命令中将使用 `helm3`）。

2. 为 Helm 3 安装 helm-2to3 插件。

   ```
   helm3 plugin install https://github.com/helm/helm-2to3
   ```

   通过如下命令可确认是否已正确安装 helm-2to3 插件。

   ```
   helm3 plugin list
   ```

   ```
   NAME   VERSION DESCRIPTION
   2to3   0.8.0  migrate and cleanup Helm v2 configuration and releases in
       ↪ -place to Helm v3
   ```

3. 迁移 Helm 2 的仓库、插件等配置到 Helm 3。

   ```
   helm3 2to3 move config
   ```

   在正式迁移配置前，可使用 `helm3 2to3 move config --dry-run` 了解可能执行的操作及其影响。

   迁移配置完成后，可看到 Helm 3 中已包含 PingCAP 仓库。

   ```
   helm3 repo list
   ```

   ```
   NAME   URL
   pingcap https://charts.pingcap.org/
   ```

4. 迁移 Helm 2 管理的 release 到 Helm 3。

   ```
   helm3 2to3 convert tidb-operator
   ```

   在正式迁移 release 前，可使用 `helm3 2to3 convert tidb-operator --dry-run` 了解可能执行的操作及其影响。

   迁移 release 完成后，可通过 Helm 3 看到 TiDB Operator 对应的 release。

   ```
   helm3 list --namespace=tidb-admin
   ```

```
NAME            NAMESPACE       REVISION        UPDATED
    ↪                                   STATUS          CHART                   APP
    ↪  VERSION
tidb-operator tidb-admin      1               2021-01-05 07:28:00.3545941
    ↪ +0000 UTC deployed    tidb-operator-v1.1.8 v1.1.8
```

> **注意：**
>
> 如果原 Helm 2 是 Tillerless 的（通过 helm-tiller 等插件将 Tiller 安装在本地而不是 Kubernetes 集群中），则可以通过增加 `--tiller-out-cluster` 参数进行迁移，即 `helm3 2to3 convert tidb-operator --tiller-out-cluster`。

5. 确认 TiDB Operator、TidbCluster 及 TidbMonitor 运行正常。

   使用以下命令检查 TiDB Operator 组件是否运行正常：

```
kubectl get pods --namespace=tidb-admin -l app.kubernetes.io/instance=
    ↪ tidb-operator
```

   期望输出为所有 Pod 都处于 Running 状态：

```
NAME                                      READY   STATUS    RESTARTS   AGE
tidb-controller-manager-6d8d5c6d64-b8lv4  1/1     Running   0          2m22s
tidb-scheduler-644d59b46f-4f6sb           2/2     Running   0          2m22s
```

   使用以下命令检查 TidbCluster 和 TidbMonitor 组件是否运行正常：

```
watch kubectl get pods --namespace=tidb-cluster
```

   期望输出为所有 Pod 都处于 Running 状态：

```
NAME                              READY   STATUS    RESTARTS   AGE
basic-discovery-6bb656bfd-xl5pb   1/1     Running   0          9m9s
basic-monitor-5fc8589c89-gvgjj    3/3     Running   0          8m58s
basic-pd-0                        1/1     Running   0          9m8s
basic-tidb-0                      2/2     Running   0          7m14s
basic-tikv-0                      1/1     Running   0          8m13s
```

6. 清理 Helm 2 对应的配置、release 信息等数据。

```
helm3 2to3 cleanup --name=tidb-operator
```

   在正式清理数据前，可使用 `helm3 2to3 cleanup --name=tidb-operator --dry-run` 了解可能执行的操作及其影响。

注意：
清理完成后，Helm 2 中将无法再管理对应的 release。

7 灾难恢复

7.1 使用 PD Recover 恢复 PD 集群

PD Recover 是对 PD 进行灾难性恢复的工具，用于恢复无法正常启动或服务的 PD 集群。

7.1.1 下载 PD Recover

1. 下载 TiDB 官方安装包：

```
wget https://download.pingcap.org/tidb-${version}-linux-amd64.tar.gz
```

`${version}` 是 TiDB 集群版本，例如，`v5.0.6`。

2. 解压安装包：

```
tar -xzf tidb-${version}-linux-amd64.tar.gz
```

`pd-recover` 在 `tidb-${version}-linux-amd64/bin` 目录下。

7.1.2 使用 PD Recover 恢复 PD 集群

本小节详细介绍如何使用 PD Recover 来恢复 PD 集群。

7.1.2.1 获取 Cluster ID

```
kubectl get tc ${cluster_name} -n ${namespace} -o='go-template={{.status.
    ↪ clusterID}}{{"\n"}}'
```

示例：

```
kubectl get tc test -n test -o='go-template={{.status.clusterID}}{{"\n"}}'
6821434242797747735
```

7.1.2.2 获取 Alloc ID

使用 pd-recover 恢复 PD 集群时，需要指定 alloc-id。alloc-id 的值需要是一个比当前已经分配的最大的 Alloc ID 更大的值。

1. 参考访问 Prometheus 监控数据打开 TiDB 集群的 Prometheus 访问页面。
2. 在输入框中输入 pd_cluster_id 并点击 Execute 按钮查询数据，获取查询结果中的最大值。
3. 将查询结果中的最大值乘以 100，作为使用 pd-recover 时指定的 alloc-id。

7.1.2.3 恢复 PD 集群 Pod

1. 删除 PD 集群 Pod。

   通过如下命令设置 spec.pd.replicas 为 0：

   ```
   kubectl edit tc ${cluster_name} -n ${namespace}
   ```

   由于此时 PD 集群异常，TiDB Operator 无法将上面的改动同步到 PD StatefulSet，所以需要通过如下命令设置 PD StatefulSet spec.replicas 为 0：

   ```
   kubectl edit sts ${cluster_name}-pd -n ${namespace}
   ```

   通过如下命令确认 PD Pod 已经被删除：

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

2. 确认所有 PD Pod 已经被删除后，通过如下命令删除 PD Pod 绑定的 PVC：

   ```
   kubectl delete pvc -l app.kubernetes.io/component=pd,app.kubernetes.io/
       ↪ instance=${cluster_name} -n ${namespace}
   ```

3. PVC 删除完成后，扩容 PD 集群至一个 Pod。

   通过如下命令设置 spec.pd.replicas 为 1：

   ```
   kubectl edit tc ${cluster_name} -n ${namespace}
   ```

   由于此时 PD 集群异常，TiDB Operator 无法将上面的改动同步到 PD StatefulSet，所以需要通过如下命令设置 PD StatefulSet spec.replicas 为 1：

   ```
   kubectl edit sts ${cluster_name}-pd -n ${namespace}
   ```

   通过如下命令确认 PD 已经启动：

   ```
   kubectl logs -f ${cluster_name}-pd-0 -n ${namespace} | grep "Welcome to
       ↪  Placement Driver (PD)"
   ```

7.1.2.4 使用 PD Recover 恢复集群

1. 通过 port-forward 暴露 PD 服务：

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379
```

2. 打开一个新终端标签或窗口，进入到 pd-recover 所在的目录，使用 pd-recover 恢复 PD 集群：

```
./pd-recover -endpoints http://127.0.0.1:2379 -cluster-id ${cluster_id}
    ↪  -alloc-id ${alloc_id}
```

${cluster_id} 是获取 Cluster ID 步骤中获取的 Cluster ID，${alloc_id} 是获取 Alloc ID 步骤中获取的 pd_cluster_id 的最大值再乘以 100。

pd-recover 命令执行成功后，会打印如下输出：

```
recover success! please restart the PD cluster
```

3. 回到 port-forward 命令所在窗口，按 Ctrl+C 停止并退出。

7.1.2.5 重启 PD Pod

1. 删除 PD Pod：

```
kubectl delete pod ${cluster_name}-pd-0 -n ${namespace}
```

2. Pod 正常启动后，通过 port-forward 暴露 PD 服务：

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379
```

3. 打开一个新终端标签或窗口，通过如下命令确认 Cluster ID 为获取 Cluster ID 步骤中获取的 Cluster ID：

```
curl 127.0.0.1:2379/pd/api/v1/cluster
```

4. 回到 port-forward 命令所在窗口，按 Ctrl+C 停止并退出。

7.1.2.6 扩容 PD 集群

通过如下命令设置 spec.pd.replicas 为期望的 Pod 数量：

```
kubectl edit tc ${cluster_name} -n ${namespace}
```

7.1.2.7 重启 TiDB 和 TiKV

```
kubectl delete pod -l app.kubernetes.io/component=tidb,app.kubernetes.io/
    ↪ instance=${cluster_name} -n ${namespace} &&
kubectl delete pod -l app.kubernetes.io/component=tikv,app.kubernetes.io/
    ↪ instance=${cluster_name} -n ${namespace}
```

7.2 恢复误删的 TiDB 集群

本文介绍了如何恢复误删的 TiDB 集群。

7.2.1 TidbCluster 管理的集群意外删除后恢复

TiDB Operator 使用 PV (Persistent Volume)、PVC (Persistent Volume Claim) 来存储持久化的数据，如果不小心使用 `kubectl delete tc` 意外删除了集群，PV/PVC 对象以及数据都会保留下来，以最大程度保证数据安全。

此时集群恢复的办法就是使用 `kubectl create` 命令来创建一个同名同配置的集群，之前保留下来未被删除的 PV/PVC 以及数据会被复用：

```
kubectl -n ${namespace} create -f tidb-cluster.yaml
```

7.2.2 Helm 管理的集群意外删除后恢复

TiDB Operator 使用 PV (Persistent Volume)、PVC (Persistent Volume Claim) 来存储持久化的数据，如果不小心使用 `helm uninstall` 意外删除了集群，PV/PVC 对象以及数据都会保留下来，以最大程度保证数据安全。

此时集群恢复的办法就是使用 `helm install` 命令来创建一个同名同配置的集群，之前保留下来未被删除的 PV/PVC 以及数据会被复用：

```
helm install ${release_name} pingcap/tidb-cluster --namespace=${namespace}
    ↪ --version=${chart_version} -f values.yaml
```

8 导入集群数据

本文介绍了如何使用 TiDB Lightning 导入集群数据。

TiDB Lightning 包含两个组件：tidb-lightning 和 tikv-importer。在 Kubernetes 上，tikv-importer 位于单独的 Helm chart 内，被部署为一个副本数为 1 (`replicas=1`) 的 `StatefulSet`；tidb-lightning 位于单独的 Helm chart 内，被部署为一个 `Job`。

目前，TiDB Lightning 支持三种后端：`Importer-backend`、`Local-backend` 、`TiDB-backend`。关于这三种后端的区别和选择，请参阅 TiDB Lightning 文档。对于 `Importer-backend` 后端，需要分别部署 tikv-importer 与 tidb-lightning；对于 `Local-backend` 或 `TiDB-backend` 后端，仅需要部署 tidb-lightning。

此外，对于 TiDB-backend 后端，推荐使用基于 TiDB Operator 新版（v1.1 及以上）的 CustomResourceDefinition (CRD) 实现。具体信息可参考使用 TiDB Lightning 恢复 GCS 上的备份数据或使用 TiDB Lightning 恢复 S3 兼容存储上的备份数据。

8.1 部署 TiKV Importer

> **注意：**
>
> 如需要使用 TiDB Lightning 的 local 或 tidb 后端用于数据恢复，则不需要部署 tikv-importer。

可以通过 tikv-importer Helm chart 来部署 tikv-importer，示例如下：

1. 确保 PingCAP Helm 库是最新的：

```
helm repo update
```

```
helm search repo tikv-importer -l
```

2. 获取默认的 values.yaml 文件以方便自定义：

```
helm inspect values pingcap/tikv-importer --version=${chart_version} >
    ↪ values.yaml
```

3. 修改 values.yaml 文件以指定目标 TiDB 集群。示例如下：

```
clusterName: demo
image: pingcap/tidb-lightning:v5.0.6
imagePullPolicy: IfNotPresent
storageClassName: local-storage
storage: 20Gi
pushgatewayImage: prom/pushgateway:v0.3.1
pushgatewayImagePullPolicy: IfNotPresent
config: |
  log-level = "info"
  [metric]
  job = "tikv-importer"
  interval = "15s"
  address = "localhost:9091"
```

clusterName 必须匹配目标 TiDB 集群。

如果目标 TiDB 集群组件间开启了 TLS (spec.tlsCluster.enabled: true)，则可以参考为 TiDB 集群各个组件生成证书为 TiKV importer 组件生成 Server 端证书，并在 values.yaml 中通过配置 tlsCluster.enabled: true 开启 TLS 支持。

4. 部署 tikv-importer：

```
helm install ${cluster_name} pingcap/tikv-importer --namespace=${
    ↪ namespace} --version=${chart_version} -f values.yaml
```

> 注意：
> tikv-importer 必须与目标 TiDB 集群安装在相同的命名空间中。

8.2 部署 TiDB Lightning

8.2.1 配置 TiDB Lightning

使用如下命令获得 TiDB Lightning 的默认配置：

```
helm inspect values pingcap/tidb-lightning --version=${chart_version} > tidb
    ↪ -lightning-values.yaml
```

根据需要配置 TiDB Lightning 所使用的后端 backend，即将 values.yaml 中的 backend 设置为 importer、local 、tidb 中的一个。

> 注意：
> 如果使用 local 后端，则还需要在 values.yaml 中设置 sortedKV 来创建相应的 PVC 以用于本地 KV 排序。

自 v1.1.10 版本起，tidb-lightning Helm chart 默认会将 TiDB Lightning 的 checkpoint 信息存储在源数据所在目录内。这样在运行新的 lightning job 时，可以根据 checkpoint 信息进行断点续传。

对于 v1.1.10 之前的版本，可参考 TiDB Lightning 断点续传，在 values.yaml 中的 config 配置下，设置将 checkpoint 信息保存到目标 TiDB 集群、其他 MySQL 协议兼容的数据库或共享存储目录中。

如果目标 TiDB 集群组件间开启了 TLS (spec.tlsCluster.enabled: true)，则可以参考为 TiDB 集群各个组件生成证书为 TiDB Lightning 组件生成 Server 端证书，并在 values.yaml 中通过配置 tlsCluster.enabled: true 开启集群内部的 TLS 支持。

如果目标 TiDB 集群为 MySQL 客户端开启了 TLS (`spec.tidb.tlsClient.enabled: true`) 并配置了相应的 Client 端证书 (对应的 Kubernetes Secret 对象为 `${cluster_name}-tidb-client-secret`)，则可以通过在 `values.yaml` 中配置 `tlsClient.enabled: true` 以使 TiDB Lightning 通过 TLS 方式连接 TiDB Server。

如果需要 TiDB Lightning 使用不同的 Client 证书来连接 TiDB Server，则可以参考为 TiDB 集群颁发两套证书为 TiDB Lightning 组件生成 Client 端证书，并在 `values.yaml` 中通过 `tlsCluster.tlsClientSecretName` 指定对应的 Kubernetes Sceret 对象。

注意：

如果通过 `tlsCluster.enabled: true` 开启了集群内部的 TLS 支持，但未通过 `tlsClient.enabled: true` 开启 TiDB Lightning 到 TiDB Server 的 TLS 支持，则需要在 `values.yaml` 中的 `config` 内通过如下配置显式地禁用 TiDB Lightning 到 TiDB Server 的 TLS 连接支持。

```
[tidb]
tls="false"
```

tidb-lightning Helm chart 支持恢复本地或远程的备份数据。

8.2.1.1 本地模式

本地模式要求备份工具导出的备份数据位于其中一个 Kubernetes 节点上。要启用该模式，你需要将 `dataSource.local.nodeName` 设置为该节点名称，将 `dataSource.local.hostPath` 设置为备份数据目录路径，该路径中需要包含名为 `metadata` 的文件。

8.2.1.2 远程模式

与本地模式不同，远程模式需要使用 rclone 将备份工具备份的 tarball 文件从网络存储中下载到 PV 中。远程模式能在 rclone 支持的任何云存储下工作，目前已经有以下存储进行了相关测试：Google Cloud Storage (GCS)、Amazon S3 和 Ceph Object Storage。

使用远程模式恢复备份数据的步骤如下：

1. 确保 `values.yaml` 中的 `dataSource.local.nodeName` 和 `dataSource.local.hostPath` 被注释掉。
2. 存储访问授权

   使用 Amazon S3 作为后端存储时，参考AWS 账号授权。在使用不同的权限授予方式时，需要使用不用的配置。

   使用 Ceph 作为存储后端时，参考通过 AccessKey 和 SecretKey 授权。

   使用 GCS 作为存储后端时，参考GCS 账号授权。

- 通过 AccessKey 和 SecretKey 授权

  1. 新建一个包含 rclone 配置的 Secret 配置文件 secret.yaml。rclone 配置示例如下。一般只需要配置一种云存储。

```
apiVersion: v1
kind: Secret
metadata:
  name: cloud-storage-secret
type: Opaque
stringData:
  rclone.conf: |
    [s3]
    type = s3
    provider = AWS
    env_auth = false
    access_key_id = ${access_key}
    secret_access_key = ${secret_key}
    region = us-east-1
    [ceph]
    type = s3
    provider = Ceph
    env_auth = false
    access_key_id = ${access_key}
    secret_access_key = ${secret_key}
    endpoint = ${endpoint}
    region = :default-placement
    [gcs]
    type = google cloud storage
    # 该服务账号必须被授予 Storage Object Viewer 角色。
    # 该内容可以通过 `cat ${service-account-file} | jq -c .`
        ↪ 命令获取。
    service_account_credentials = ${
        ↪ service_account_json_file_content}
```

  2. 运行以下命令创建 Secret：

```
kubectl apply -f secret.yaml -n ${namespace}
```

- 通过 IAM 绑定 Pod 授权或者通过 IAM 绑定 ServiceAccount 授权

  使用 Amazon S3 作为存储后端时支持通过 IAM 绑定 Pod 授权或者通过 IAM 绑定 ServiceAccount 授权，此时可省略 s3.access_key_id 以及 s3.secret_access_key。

  1. 将下面文件存储为 secret.yaml。

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: cloud-storage-secret
type: Opaque
stringData:
  rclone.conf: |
    [s3]
    type = s3
    provider = AWS
    env_auth = true
    access_key_id =
    secret_access_key =
    region = us-east-1
```

2. 运行以下命令创建 Secret：

```
kubectl apply -f secret.yaml -n ${namespace}
```

3. 将 dataSource.remote.storageClassName 设置为 Kubernetes 集群中现有的一个存储类型。

8.2.1.3 Ad hoc 模式

当使用远程模式进行恢复时，如果在恢复过程中由于异常而造成中断、但又不希望重复从网络存储中下载备份数据，则可以使用 Ad hoc 模式直接恢复已通过远程模式下载并解压到 PV 中的数据。步骤如下：

1. 确保 values.yaml 中的 dataSource.local 和 dataSource.remote 均为空配置。
2. 配置 values.yaml 中的 dataSource.adhoc.pvcName 为使用远程模式时创建的 PVC 名称。
3. 配置 values.yaml 中的 dataSource.adhoc.backupName 为原备份数据对应的名称，如 backup-2020-12-17T10:12:51Z (不包含在网络存储上压缩文件名的 .tgz 后缀)。

8.2.2 部署 TiDB Lightning

部署 TiDB Lightning 的方式根据不同的权限授予方式及存储方式，有不同的情况。

- 对于本地模式、Ad hoc 模式、远程模式 (需要是符合以下三个条件之一的远程模式：使用 Amazon S3 AccessKey 和 SecretKey 权限授予方式、使用 Ceph 作为存储后端、使用 GCS 作为存储后端)，运行以下命令部署 TiDB Lightning：

```
helm install ${release_name} pingcap/tidb-lightning --namespace=${
    ↪ namespace} --set failFast=true -f tidb-lightning-values.yaml --
    ↪ version=${chart_version}
```

- 使用 Amazon S3 IAM 绑定 Pod 的授权方式的远程模式时，需要完成以下步骤：

  1. 创建 IAM 角色：
     可以参考 AWS 官方文档来为账号创建一个 IAM 角色，并且通过 AWS 官方文档为 IAM 角色赋予需要的权限。由于 Lightning 需要访问 AWS 的 S3 存储，所以这里给 IAM 赋予了 AmazonS3FullAccess 的权限。
  2. 修改 tidb-lightning-values.yaml，找到 annotations 字段，增加 annotation iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user。
  3. 部署 Tidb-Lightning：

```
helm install ${release_name} pingcap/tidb-lightning --namespace=${
    ↪ namespace} --set failFast=true -f tidb-lightning-values.yaml
    ↪  --version=${chart_version}
```

> 注意：
> arn:aws:iam::123456789012:role/user 为步骤 1 中创建的 IAM 角色。

- 使用 Amazon S3 IAM 绑定 ServiceAccount 授权方式的远程模式时：

  1. 在集群上为服务帐户启用 IAM 角色：
     可以参考 AWS 官方文档 开启所在的 EKS 集群的 IAM 角色授权。
  2. 创建 IAM 角色：
     可以参考 AWS 官方文档创建一个 IAM 角色，为角色赋予 AmazonS3FullAccess 的权限，并且编辑角色的 Trust relationships。
  3. 绑定 IAM 到 ServiceAccount 资源上：

```
kubectl annotate sa ${servieaccount} -n ${namespace} eks.amazonaws.
    ↪ com/role-arn=arn:aws:iam::123456789012:role/user
```

  4. 部署 Tidb-Lightning：

```
helm install ${release_name} pingcap/tidb-lightning --namespace=${
    ↪ namespace} --set-string failFast=true,serviceAccount=${
    ↪ servieaccount} -f tidb-lightning-values.yaml --version=${
    ↪ chart_version}
```

> 注意：
> arn:aws:iam::123456789012:role/user 为步骤 1 中创建的 IAM 角色。${service-account} 为 tidb-lightning 使用的 ServiceAccount，默认为 default。

8.3 销毁 TiKV Importer 和 TiDB Lightning

目前，TiDB Lightning 只能在线下恢复数据。当恢复过程结束、TiDB 集群需要向外部应用提供服务时，可以销毁 TiDB Lightning 以节省开支。

删除 tikv-importer 的步骤：

- 运行 `helm uninstall ${release_name} -n ${namespace}`。

删除 tidb-lightning 的方法：

- 运行 `helm uninstall ${release_name} -n ${namespace}`。

8.4 故障诊断

当 TiDB Lightning 未能成功恢复数据时，通常不能简单地直接重启进程，必须进行手动干预，否则将很容易出现错误。因此，tidb-lightning 的 `Job` 重启策略被设置为 `Never`。

> 注意：
>
> 如未设置将 checkpoint 信息持久化保存到目标 TiDB 集群、其他 MySQL 协议兼容的数据库或共享存储目录中，发生故障后，需要清理目标集群中已恢复的部分数据，并重新部署 tidb-lightning 进行数据恢复。

如果 TiDB Lightning 未能成功恢复数据，且已配置将 checkpoint 信息存储在源数据所在目录、其他用户配置的数据库或存储目录中，可采用以下步骤进行手动干预：

1. 运行 `kubectl logs -n ${namespace} ${pod_name}` 查看 log。

   如果使用远程模式进行数据恢复，且异常发生在从网络存储下载数据的过程中，则依据 log 信息进行处理后，直接重新部署 tidb-lightning 进行数据恢复。否则，继续按下述步骤进行处理。

2. 依据 log 并参考 TiDB Lightning 故障排除指南，了解各故障类型的处理方法。

3. 对于不同的故障类型，分别进行处理：

   - 如果需要使用 tidb-lightning-ctl 进行处理：

     1. 设置 `values.yaml` 的 `dataSource` 以确保新 `Job` 将使用发生故障的 `Job` 已有的数据源及 checkpoint 信息：
        - 如果使用本地模式或 Ad hoc 模式，则 `dataSource` 无需修改。

  - 如果使用远程模式，则修改 dataSource 为 Ad hoc 模式。其中 dataSource.adhoc.pvcName 为原 Helm chart 创建的 PVC 名称，dataSource.adhoc.backupName 为待恢复数据的 backup 名称。
2. 修改 values.yaml 中的 failFast 为 false 并创建用于使用 tidb-lightning-ctl 的 Job。
  - TiDB Lightning 会依据 checkpoint 信息检测前一次数据恢复是否发生错误，当检测到错误时会自动中止运行。
  - TiDB Lightning 会依据 checkpoint 信息来避免对已恢复数据的重复恢复，因此创建该 Job 不会影响数据正确性。
3. 当新 Job 对应的 pod 运行后，使用 kubectl logs -n ${namespace} ${pod_name} 查看 log 并确认新 Job 中的 tidb-lightning 已停止进行数据恢复，即 log 中包含类似以下的任意信息：
  - tidb lightning encountered error
  - tidb lightning exit
4. 执行 kubectl exec -it -n ${namespace} ${pod_name} -it -- sh 命令进入容器。
5. 运行 cat /proc/1/cmdline，获得启动脚本。
6. 根据启动脚本中的命令行参数，参考 TiDB Lightning 故障排除指南并使用 tidb-lightning-ctl 进行故障处理。
7. 故障处理完成后，将 values.yaml 中的 failFast 设置为 true 并再次创建新的 Job 用于继续数据恢复。

- 如果不需要使用 tidb-lightning-ctl 进行处理：
  1. 参考 TiDB Lightning 故障排除指南进行故障处理。
  2. 设置 values.yaml 的 dataSource 以确保新 Job 将使用发生故障的 Job 已有的数据源及 checkpoint 信息：
    - 如果使用本地模式或 Ad hoc 模式，则 dataSource 无需修改。
    - 如果使用远程模式，则修改 dataSource 为 Ad hoc 模式。其中 dataSource.adhoc.pvcName 为原 Helm chart 创建的 PVC 名称，dataSource.adhoc.backupName 为待恢复数据的 backup 名称。
  3. 根据新的 values.yaml 创建新的 Job 用于继续数据恢复。

4. 故障处理及数据恢复完成后，参考销毁 TiKV Importer 和 TiDB Lightning 删除用于数据恢复的 Job 及用于故障处理的 Job。

9 故障诊断

9.1 Kubernetes 上的 TiDB 集群管理常用使用技巧

本文介绍了 Kubernetes 上 TiDB 集群管理常用使用技巧。

9.1.1 诊断模式

当 Pod 处于 CrashLoopBackoff 状态时，Pod 内容器不断退出，导致无法正常使用 kubectl exec，给诊断带来不便。为了解决这个问题，TiDB in Kubernetes 提供了 PD/TiKV/TiDB Pod 诊断模式。在诊断模式下，Pod 内的容器启动后会直接挂起，不会再进入重复 Crash 的状态，此时，便可以通过 kubectl exec 连接 Pod 内的容器进行诊断。

操作方式：

1. 首先，为待诊断的 Pod 添加 Annotation：

```
kubectl annotate pod ${pod_name} -n ${namespace} runmode=debug
```

在 Pod 内的容器下次重启时，会检测到该 Annotation，进入诊断模式。

2. 等待 Pod 进入 Running 状态即可开始诊断：

```
watch kubectl get pod ${pod_name} -n ${namespace}
```

下面是使用 kubectl exec 进入容器进行诊断工作的例子：

```
kubectl exec -it ${pod_name} -n ${namespace} -- /bin/sh
```

3. 诊断完毕，修复问题后，删除 Pod：

```
kubectl delete pod ${pod_name} -n ${namespace}
```

Pod 重建后会自动回到正常运行模式。

9.2 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。

9.2.1 Pod 未正常创建

创建集群后，如果 Pod 没有创建，则可以通过以下方式进行诊断：

```
kubectl get tidbclusters -n ${namespace}
kubectl describe tidbclusters -n ${namespace} ${cluster_name}
kubectl get statefulsets -n ${namespace}
kubectl describe statefulsets -n ${namespace} ${cluster_name}-pd
```

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 backupschedules -n ${namespace} ${backupschedule_name}
kubectl describe jobs -n ${namespace} ${backupjob_name}
kubectl describe restores -n ${namespace} ${restore_name}
```

9.2.2 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、Pump、Monitor、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
- PD 或者 TiKV Replicas 数量和集群内节点数量不满足 tidb-scheduler 高可用调度策略

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}
```

9.2.2.1 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。

9.2.2.2 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：

- 如果是启动 tidbcluster 集群，运行 `kubectl edit tc ${cluster_name} -n ${namespace}` 进行集群更新。
- 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. 将 Statefulset 删除，并且将对应的 PVC 也都删除。

```
kubectl delete pvc -n ${namespace} ${pvc_name}
kubectl delete sts -n ${namespace} ${statefulset_name}
```

9.2.2.3 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。对于 Local PV，可以参考本地 PV 配置进行扩充。

9.2.2.4 不满足 tidb-scheduler 高可用策略

tidb-scheduler 针对 PD 和 TiKV 定制了高可用调度策略。对于同一个 TiDB 集群，假设 PD 或者 TiKV 的 Replicas 数量为 N，那么可以调度到每个节点的 PD Pod 数量最多为 `M=(N-1)/2`（如果 N<3，M=1），可以调度到每个节点的 TiKV Pod 数量最多为 `M=ceil(N/3)`（ceil 表示向上取整，如果 N<3，M=1）。

如果 Pod 因为不满足高可用调度策略而导致状态为 Pending，需要往集群内添加节点。

9.2.3 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。

9.2.3.1 查看 Pod 内当前容器的日志

```
kubectl -n ${namespace} logs -f ${pod_name}
```

9.2.3.2 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。

9.2.3.3 cluster id mismatch

若是 TiKV Pod 日志中出现 “cluster id mismatch” 信息，则 TiKV Pod 使用的数据可能是其他或之前的 TiKV Pod 的旧数据。在集群配置本地存储时未清除机器上本地磁盘上的数据，或者强制删除了 PV 导致数据并没有被 local volume provisioner 程序回收，可能导致 PV 遗留旧数据，导致错误。

在确认该 TiKV 应作为新节点加入集群、且 PV 上的数据应该删除后，可以删除该 TiKV Pod 和关联 PVC。TiKV Pod 将自动重建并绑定新的 PV 来使用。集群本地存储配置中，应对机器上的本地存储删除，避免 Kubernetes 使用机器上遗留的数据。集群运维中，不可强制删除 PV，应由 local volume provisioner 程序管理。用户通过创建、删除 PVC 以及设置 PV 的 reclaimPolicy 来管理 PV 的生命周期。

9.2.3.4 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
```

9.2.3.5 其他原因

假如通过日志无法确认失败原因，ulimit 也设置正常，那么可以通过诊断模式进行进一步排查。

9.3 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。

9.3.1 TiKV Store 异常进入 Tombstone 状态

正常情况下，当 TiKV Pod 处于健康状态时（Pod 状态为 `Running`），对应的 TiKV Store 状态也是健康的（Store 状态为 `UP`）。但并发进行 TiKV 组件的扩容和缩容可能会导致部分 TiKV Store 异常并进入 Tombstone 状态。此时，可以按照以下步骤进行修复：

1. 查看 TiKV Store 状态：

```
kubectl get -n ${namespace} tidbcluster ${cluster_name} -ojson | jq '.
    ↪ status.tikv.stores'
```

2. 查看 TiKV Pod 运行状态：

```
kubectl get -n ${namespace} po -l app.kubernetes.io/component=tikv
```

3. 对比 Store 状态与 Pod 运行状态。假如某个 TiKV Pod 所对应的 Store 处于 `Offline` 状态，则表明该 Pod 的 Store 正在异常下线中。此时，可以通过下面的命令取消下线进程，进行恢复：

    1. 打开到 PD 服务的连接：

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-pd ${
    ↪ local_port}:2379 &>/tmp/portforward-pd.log &
```

    2. 上线对应 Store：

```
curl -X POST http://127.0.0.1:2379/pd/api/v1/store/${store_id}/
    ↪ state?state=Up
```

4. 假如某个 TiKV Pod 所对应的 `lastHeartbeatTime` 最新的 Store 处于 `Tombstone` 状态，则表明异常下线已经完成。此时，需要重建 Pod 并绑定新的 PV 进行恢复：

    1. 将该 Store 对应 PV 的 `reclaimPolicy` 调整为 `Delete`：

    ```
    kubectl patch $(kubectl get pv -l app.kubernetes.io/instance=${
        ↪ cluster_name},tidb.pingcap.com/store-id=${store_id} -o name)
        ↪  -p '{"spec":{"persistentVolumeReclaimPolicy":"Delete"}}
    ```

    2. 删除 Pod 使用的 PVC：

    ```
    kubectl delete -n ${namespace} pvc tikv-${pod_name} --wait=false
    ```

    3. 删除 Pod，等待 Pod 重建：

    ```
    kubectl delete -n ${namespace} pod ${pod_name}
    ```

    Pod 重建后，会以在集群中注册一个新的 Store，恢复完成。

9.3.2 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.*`，请为 `kubelet` 配置该参数，并按如下方式配置 TiDB：

```
tidb:
  ...
  podSecurityContext:
    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.*`，请按如下方式配置 TiDB：

```
tidb:
  annotations:
    tidb.pingcap.com/sysctl-init: "true"
  podSecurityContext:
    sysctls:
    - name: net.ipv4.tcp_keepalive_time
      value: "300"
  ...
```

注意：

进行以上配置要求 TiDB Operator 1.1 及以上版本。

9.4 Kubernetes 上的 TiDB 集群常见网络问题

本文介绍了 Kubernetes 上 TiDB 集群常见网络问题以及诊断解决方案。

9.4.1 Pod 之间网络不通

针对 TiDB 集群而言，绝大部分 Pod 间的访问均通过 Pod 的域名（使用 Headless Service 分配）进行，例外的情况是 TiDB Operator 在收集集群信息或下发控制指令时，会通过 PD Service 的 `service-name` 访问 PD 集群。

当通过日志或监控确认 Pod 间存在网络连通性问题，或根据故障情况推断出 Pod 间网络连接可能不正常时，可以按照下面的流程进行诊断，逐步缩小问题范围：

1. 确认 Service 和 Headless Service 的 Endpoints 是否正常：

```
kubectl -n ${namespace} get endpoints ${cluster_name}-pd
kubectl -n ${namespace} get endpoints ${cluster_name}-tidb
kubectl -n ${namespace} get endpoints ${cluster_name}-pd-peer
kubectl -n ${namespace} get endpoints ${cluster_name}-tikv-peer
kubectl -n ${namespace} get endpoints ${cluster_name}-tidb-peer
```

以上命令展示的 `ENDPOINTS` 字段中，应当是由逗号分隔的 `cluster_ip:port` 列表。假如字段为空或不正确，请检查 Pod 的健康状态以及 `kube-controller-manager` 是否正常工作。

2. 进入 Pod 的 Network Namespace 诊断网络问题：

```
tkctl debug -n ${namespace} ${pod_name}
```

远端 shell 启动后，使用 dig 命令诊断 DNS 解析，假如 DNS 解析异常，请参照诊断 Kubernetes DNS 解析进行故障排除：

```
dig ${HOSTNAME}
```

使用 ping 命令诊断到目的 IP 的三层网络是否连通（目的 IP 为使用 dig 解析出的 Pod IP）：

```
ping ${TARGET_IP}
```

假如 ping 检查失败，请参照诊断 Kubernetes 网络进行故障排除。

假如 ping 检查正常，继续使用 telnet 检查目标端口是否打开：

```
telnet ${TARGET_IP} ${TARGET_PORT}
```

假如 telnet 检查失败，则需要验证 Pod 的对应端口是否正确暴露以及应用的端口是否配置正确：

```
# 检查端口是否一致
kubectl -n ${namespace} get po ${pod_name} -ojson | jq '.spec.
    ↪ containers[].ports[].containerPort'

# 检查应用是否被正确配置服务于指定端口上
# PD，未配置时默认为 2379 端口
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/pd/pd.toml |
    ↪ grep client-urls
# TiKV，未配置时默认为 20160 端口
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/tikv/tikv.toml
    ↪  | grep addr
# TiDB，未配置时默认为 4000 端口
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/tidb/tidb.toml
    ↪  | grep port
```

9.4.2 无法访问 TiDB 服务

TiDB 服务访问不了时，首先确认 TiDB 服务是否部署成功，确认方法如下：

查看该集群的所有组件是否全部都启动了，状态是否为 Running。

```
kubectl get po -n ${namespace}
```

查看 TiDB 服务是否正确生成了 Endpoint 对象。

```
kubectl get endpoints -n ${namespaces} ${cluster_name}-tidb
```

检查 TiDB 组件的日志，看日志是否有报错。

```
kubectl logs -f ${pod_name} -n ${namespace} -c tidb
```

如果确定集群部署成功，则进行网络检查：

1. 如果你是通过 NodePort 方式访问不了 TiDB 服务，请在 node 上尝试使用 clusterIP 访问 TiDB 服务，假如 clusterIP 的方式能访问，基本判断 Kubernetes 集群内的网络是正常的，问题可能出在下面两个方面：

    - 客户端到 node 节点的网络不通。
    - 查看 TiDB service 的 externalTrafficPolicy 属性是否为 Local。如果是 Local 则客户端必须通过 TiDB Pod 所在 node 的 IP 来访问。

2. 如果 clusterIP 方式也访问不了 TiDB 服务，尝试用 TiDB 服务后端的 <PodIP>:4000 连接看是否可以访问，如果通过 PodIP 可以访问 TiDB 服务，可以确认问题出在 clusterIP 到 PodIP 之间的连接上，排查项如下：

    - 检查各个 node 上的 kube-proxy 是否正常运行：

    ```
    kubectl get po -n kube-system -l k8s-app=kube-proxy
    ```

    - 检查 node 上的 iptables 规则中 TiDB 服务的规则是否正确：

    ```
    iptables-save -t nat |grep ${clusterIP}
    ```

    - 检查对应的 endpoint 是否正确：

    ```
    kubectl get endpoints -n ${namespaces} ${cluster_name}-tidb
    ```

3. 如果通过 PodIP 访问不了 TiDB 服务，问题出在 Pod 层面的网络上，排查项如下：

    - 检查 node 上的相关 route 规则是否正确
    - 检查网络插件服务是否正常
    - 参考上面的Pod 之间网络不通章节

10 Kubernetes 上的 TiDB 集群常见问题

本文介绍 Kubernetes 上的 TiDB 集群常见问题以及解决方案。

10.1 如何修改时区设置？

默认情况下，在 Kubernetes 集群上部署的 TiDB 集群各组件容器中的时区为 UTC，如果要修改时区配置，有下面两种情况：

10.1.1 第一次部署集群

配置 TidbCluster CR 的 .spec.timezone 属性，例如：

```
...
spec:
  timezone: Asia/Shanghai
...
```

然后部署 TiDB 集群。

10.1.2 集群已经在运行

如果 TiDB 集群已经在运行，需要先升级 TiDB 集群，然后再配置 TiDB 集群支持新的时区。

1. 升级 TiDB 集群

   配置 TidbCluster CR 的 .spec.timezone 属性，例如：

   ```
   ...
   spec:
     timezone: Asia/Shanghai
   ...
   ```

   然后升级 TiDB 集群。

2. 修改 TiDB 支持新的时区

   参考时区支持，修改 TiDB 服务时区配置。

10.2 TiDB 相关组件可以配置 HPA 或 VPA 么？

TiDB 集群目前还不支持 HPA (Horizontal Pod Autoscaling，自动水平扩缩容) 和 VPA (Vertical Pod Autoscaling，自动垂直扩缩容)，因为对于数据库这种有状态应用而言，实现自动扩缩容难度较大，无法仅通过 CPU 和 memory 监控数据来简单地实现。

10.3 使用 TiDB Operator 编排 TiDB 集群时，有什么场景需要人工介入操作吗？

如果不考虑 Kubernetes 集群本身的运维，TiDB Operator 存在以下可能需要人工介入的场景：

- TiKV 自动故障转移之后的集群调整，参考自动故障转移；
- 维护或下线指定的 Kubernetes 节点，参考维护节点。

10.4 在公有云上使用 TiDB Operator 编排 TiDB 集群时，推荐的部署拓扑是怎样的？

首先，为了实现高可用和数据安全，我们在推荐生产环境下的 TiDB 集群中至少部署在三个可用区 (Available Zone)。

当考虑 TiDB 集群与业务服务的部署拓扑关系时，TiDB Operator 支持下面几种部署形态。它们有各自的优势与劣势，具体选型需要根据实际业务需求进行权衡：

- 将 TiDB 集群与业务服务部署在同一个 VPC 中的同一个 Kubernetes 集群上；
- 将 TiDB 集群与业务服务部署在同一个 VPC 中的不同 Kubernetes 集群上；
- 将 TiDB 集群与业务服务部署在不同 VPC 中的不同 Kubernetes 集群上。

10.5 TiDB Operator 支持 TiSpark 吗？

TiDB Operator 尚不支持自动编排 TiSpark。

假如要为 TiDB in Kubernetes 添加 TiSpark 组件，你需要在同一个 Kubernetes 集群中自行维护 Spark，确保 Spark 能够访问到 PD 和 TiKV 实例的 IP 与端口，并为 Spark 安装 TiSpark 插件，TiSpark 插件的安装方式可以参考 TiSpark。

在 Kubernetes 上维护 Spark 可以参考 Spark 的官方文档：Spark on Kubernetes。

10.6 如何查看 TiDB 集群配置？

如果需要查看当前集群的 PD、TiKV、TiDB 组件的配置信息，可以执行下列命令:

- 查看 PD 配置文件

```
kubectl exec -it ${pod_name} -n ${namespace} -- cat /etc/pd/pd.toml
```

- 查看 TiKV 配置文件

```
kubectl exec -it ${pod_name} -n ${namespace} -- cat /etc/tikv/tikv.toml
```

- 查看 TiDB 配置文件

```
kubectl exec -it ${pod_name} -c tidb -n ${namespace} -- cat /etc/tidb/
    ↪ tidb.toml
```

10.7 部署 TiDB 集群时调度失败是什么原因？

TiDB Operator 调度 Pod 失败的原因可能有三种情况：

- 资源不足，导致 Pod 一直阻塞在 `Pending` 状态。详细说明参见集群故障诊断。
- 部分 Node 被打了 `taint`，导致 Pod 无法调度到对应的 Node 上。详请参考 taint & toleration。
- 调度错误，导致 Pod 一直阻塞在 `ContainerCreating` 状态。这种情况发生时请检查 Kubernetes 集群中是否部署过多个 TiDB Operator。重复的 TiDB Operator 自定义调度器的存在，会导致同一个 Pod 的调度周期不同阶段会分别被不同的调度器处理，从而产生冲突。

  执行以下命令，如果有两条及以上记录，就说明 Kubernetes 集群中存在重复的 TiDB Operator，请根据实际情况删除多余的 TiDB Operator。

```
kubectl get deployment --all-namespaces |grep tidb-scheduler
```

10.8 TiDB 如何保证数据安全可靠？

TiDB Operator 部署的 TiDB 集群使用 Kubernetes 集群提供的持久卷作为存储，保证数据的持久化存储。

PD 和 TiKV 使用 Raft 一致性算法将存储的数据在各节点间复制为多副本，以确保某个节点宕机时数据的安全性。

在底层，TiKV 使用复制日志 + 状态机 (State Machine) 的模型来复制数据。对于写入请求，数据被写入 Leader，然后 Leader 以日志的形式将命令复制到它的 Follower 中。当集群中的大多数节点收到此日志时，日志会被提交，状态机会相应作出变更。

11 参考

11.1 架构

11.1.1 TiDB Operator 架构

本文档介绍 TiDB Operator 的架构及其工作原理。

11.1.1.1 架构

下图是 TiDB Operator 的架构概览。

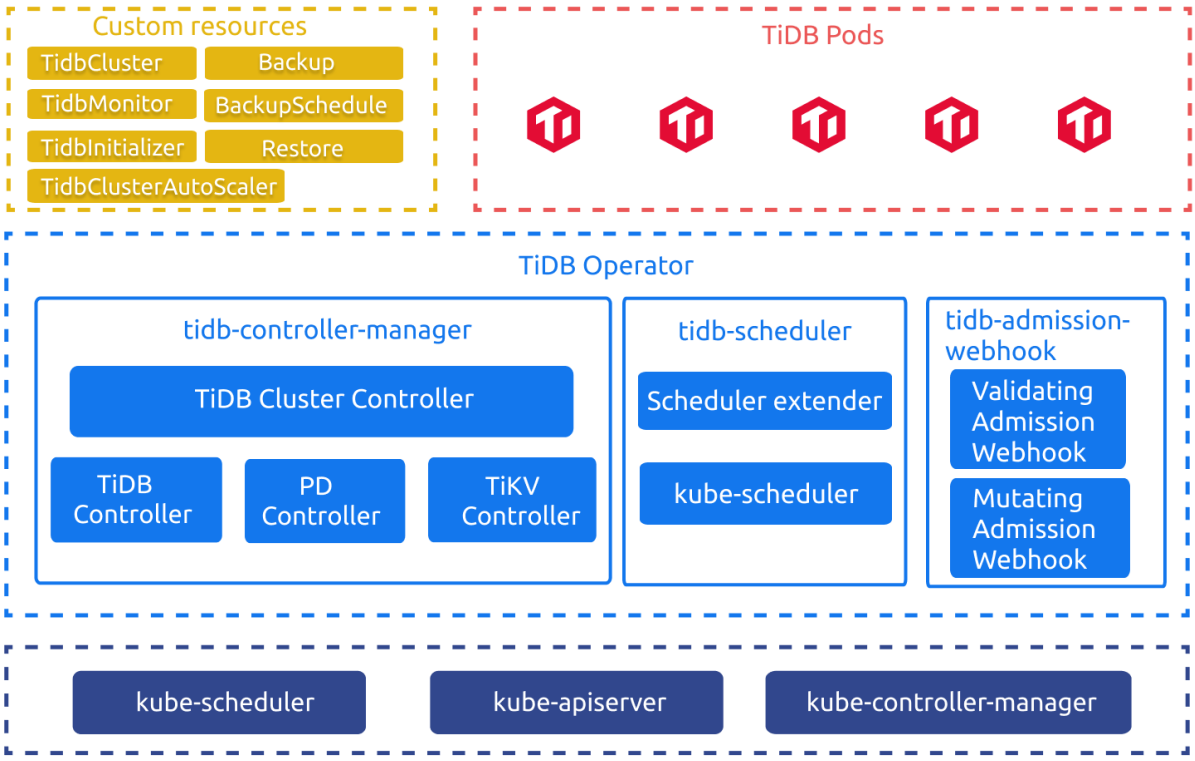


Figure 1: TiDB Operator Overview

其中，TidbCluster、TidbMonitor、TidbInitializer、Backup、Restore、BackupSchedule、TidbClusterAutoScaler 是由 CRD（CustomResourceDefinition）定义的自定义资源：

- TidbCluster 用于描述用户期望的 TiDB 集群
- TidbMonitor 用于描述用户期望的 TiDB 集群监控组件
- TidbInitializer 用于描述用户期望的 TiDB 集群初始化 Job
- Backup 用于描述用户期望的 TiDB 集群备份
- Restore 用于描述用户期望的 TiDB 集群恢复
- BackupSchedule 用于描述用户期望的 TiDB 集群周期性备份
- TidbClusterAutoScaler 用于描述用户期望的 TiDB 集群自动伸缩

TiDB 集群的编排和调度逻辑则由下列组件负责：

- tidb-controller-manager 是一组 Kubernetes 上的自定义控制器。这些控制器会不断对比 TidbCluster 对象中记录的期望状态与 TiDB 集群的实际状态，并调整 Kubernetes 中的资源以驱动 TiDB 集群满足期望状态，并根据其他 CR 完成相应的控制逻辑；

- `tidb-scheduler` 是一个 Kubernetes 调度器扩展，它为 Kubernetes 调度器注入 TiDB 集群特有的调度逻辑。
- `tidb-admission-webhook` 是一个 Kubernetes 动态准入控制器，完成 Pod、StatefulSet 等相关资源的修改、验证与运维。

此外，TiDB Operator 还提供了命令行接口 `tkctl` 用于运维集群和诊断集群问题。

11.1.1.2 流程解析

下图是 TiDB Operator 的控制流程解析。从 TiDB Operator v1.1 开始，TiDB 集群、监控、初始化、备份等组件，都通过 CR 进行部署、管理。

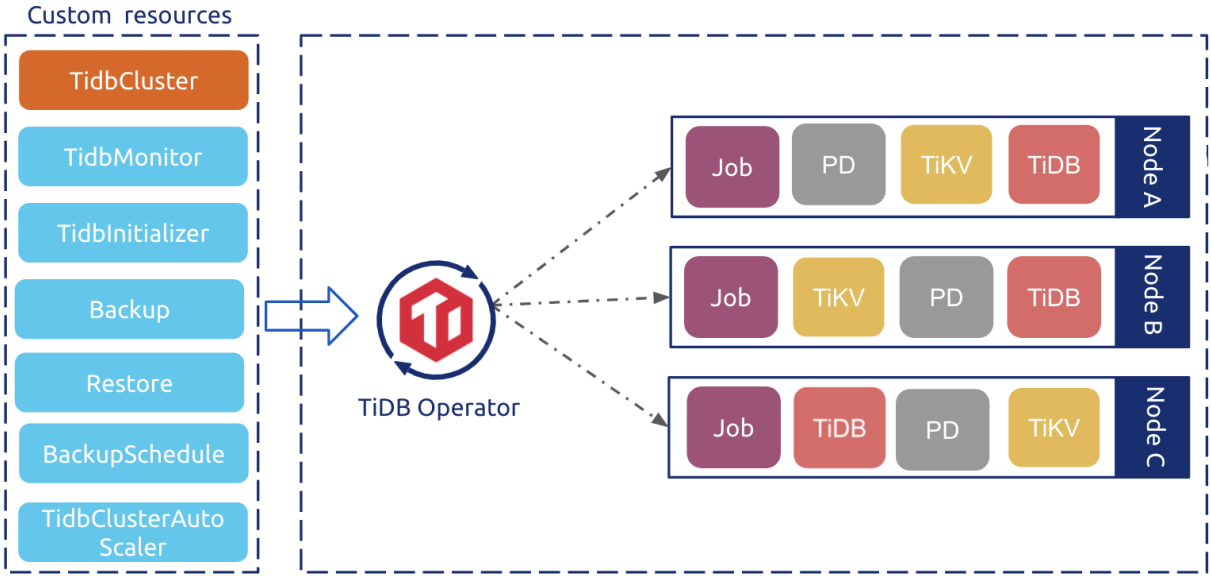


Figure 2: TiDB Operator Control Flow

整体的控制流程如下：

1. 用户通过 kubectl 创建 `TidbCluster` 和其他 CR 对象，比如 `TidbMonitor` 等；
2. TiDB Operator 会 watch `TidbCluster` 以及其它相关对象，基于集群的实际状态不断调整 PD、TiKV、TiDB 或者 Monitor 等组件的 `StatefulSet`、`Deployment` 和 `Service` 等对象；
3. Kubernetes 的原生控制器根据 `StatefulSet`、`Deployment`、`Job` 等对象创建更新或删除对应的 `Pod`；
4. PD、TiKV、TiDB 的 `Pod` 声明中会指定使用 `tidb-scheduler` 调度器，`tidb-scheduler` 会在调度对应 `Pod` 时应用 TiDB 的特定调度逻辑。

基于上述的声明式控制流程，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 `TidbCluster` 对象声明“一键”完成。

11.1.2 TiDB Scheduler 扩展调度器

TiDB Scheduler 是 Kubernetes 调度器扩展 的 TiDB 实现。TiDB Scheduler 用于向 Kubernetes 添加新的调度规则。本文介绍 TiDB Scheduler 扩展调度器的工作原理。

11.1.2.1 TiDB 集群调度需求

TiDB 集群包括 PD，TiKV 以及 TiDB 三个核心组件，每个组件又是由多个节点组成，PD 是一个 Raft 集群，TiKV 是一个多 Raft Group 集群，并且这两个组件都是有状态的。默认 Kubernetes 的调度器的调度规则无法满足 TiDB 集群的高可用调度需求，需要扩展 Kubernetes 的调度规则。

目前，可通过修改 TidbCluster 的 `metadata.annotations` 来按照特定的维度进行调度，比如：

```
metadata:
  annotations:
    pingcap.com/ha-topology-key: kubernetes.io/hostname
```

或者修改 tidb-cluster chart 的 `values.yaml`：

```
haTopologyKey: kubernetes.io/hostname
```

上述配置按照节点（默认值）维度进行调度，若要按照其他维度调度，比如: `pingcap.com/ha-topology-key: zone`，表示按照 zone 调度，还需给各节点打如下标签：

```
kubectl label nodes node1 zone=zone1
```

不同节点可有不同的标签，也可有相同的标签，如果某一个节点没有打该标签，则调度器不会调度 pod 到该节点。

目前，TiDB Scheduler 实现了如下几种自定义的调度规则（下述示例基于节点调度，基于其他维度的调度规则相同）。

11.1.2.1.1 PD 组件

调度规则一：确保每个节点上调度的 PD 实例个数小于 `Replicas / 2`。例如：

PD 集群规模（Replicas）	每个节点最多可调度的 PD 实例数量
1	1
2	1
3	1
4	1
5	2
…	

11.1.2.1.2 TiKV 组件

调度规则二：如果 Kubernetes 节点数小于 3 个（Kubernetes 集群节点数小于 3 个是无法实现 TiKV 高可用的），则可以任意调度；否则，每个节点上可调度的 TiKV 个数的计算公式为：`ceil(Replicas/3)` 。例如：

TiKV 集群规模（Replicas）	每个节点最多可调度的 TiKV 实例数量	最佳调度分布
3	1	1，1，1
4	2	1，1，2
5	2	1，2，2
6	2	2，2，2
7	3	2，2，3
8	3	2，3，3
…		

11.1.2.1.3 TiDB 组件

调度规则三：在 TiDB 实例滚动更新的时候，尽量将其调度回原来的节点。

这样实现了稳定调度，对于手动将 Node IP + NodePort 挂载在 LB 后端的场景比较有帮助，避免升级集群后 Node IP 发生变更时需要重新调整 LB，这样可以减少滚动更新时对集群的影响。

11.1.2.2 工作原理

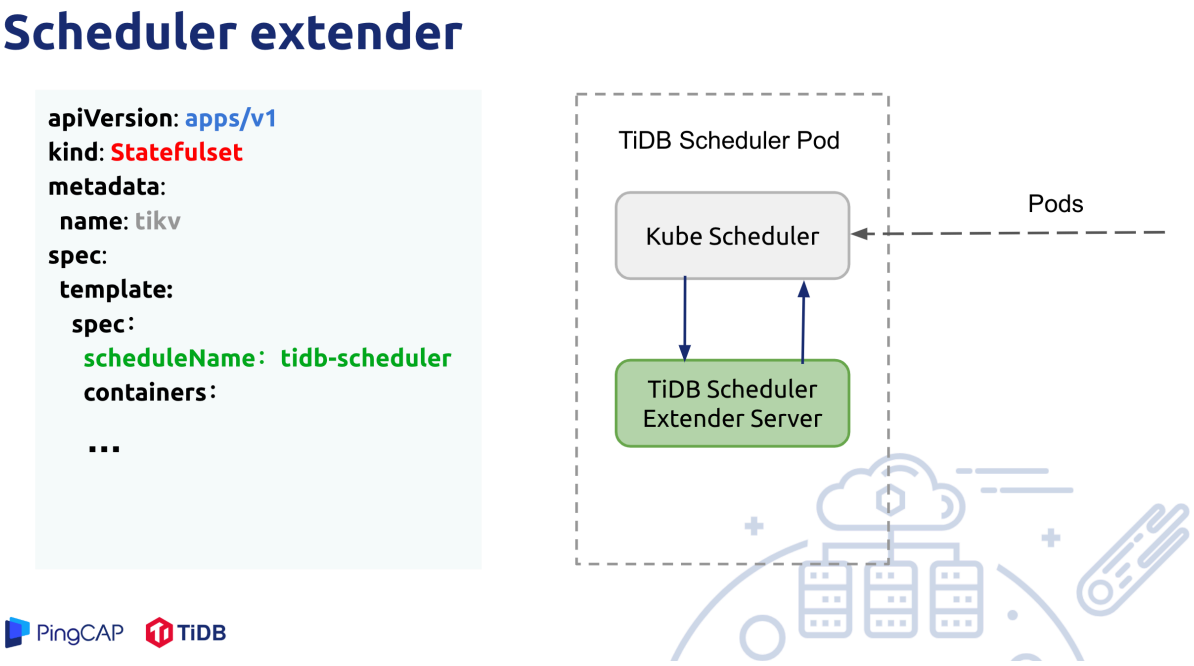


Figure 3: TiDB Scheduler 工作原理

TiDB Scheduler 通过实现 Kubernetes 调度器扩展（Scheduler extender）来添加自定义调度规则。

TiDB Scheduler 组件部署为一个或者多个 Pod，但同时只有一个 Pod 在工作。Pod 内部有两个 Container，一个 Container 是原生的 `kube-scheduler`；另外一个 Container 是 `tidb-scheduler`，实现为一个 Kubernetes scheduler extender。

TiDB Operator 创建的 PD、TiDB、TiKV Pod 的 `.spec.schedulerName` 属性会被设置为 `tidb-scheduler`，即都用 TiDB Scheduler 自定义调度器来调度。如果是测试集群，并且不要求高可用，可以将 `.spec.schedulerName` 改成 `default-scheduler` 使用 Kubernetes 内置的调度器。

一个 Pod 的调度流程是这样的：

- `kube-scheduler` 拉取所有 `.spec.schedulerName` 为 `tidb-scheduler` 的 Pod，对于每个 Pod 会首先经过 Kubernetes 默认调度规则过滤；
- 在这之后，`kube-scheduler` 会发请求到 `tidb-scheduler` 服务，`tidb-scheduler` 会通过一些自定义的调度规则（见上述介绍）对发送过来的节点进行过滤，最终将剩余可调度的节点返回给 `kube-scheduler`；
- 最终由 `kube-scheduler` 决定最终调度的节点。

如果出现 Pod 无法调度，请参考此文档诊断和解决。

11.1.3 增强型 StatefulSet 控制器

特性状态: Alpha

Kubernetes 内置 StatefulSet 为 Pods 分配连续的序号。比如 3 个副本时，Pods 分别为 pod-0, pod-1, pod-2。扩缩容时，必须在尾部增加或删除 Pods。比如扩容到 4 个副本时，会新增 pod-3。缩容到 2 副本时，会删除 pod-2。

在使用本地存储时，Pods 与 Nodes 存储资源绑定，无法自由调度。若希望删除掉中间某个 Pod，以便维护其所在的 Node 但并没有其他 Node 可以迁移时，或者某个 Pod 故障想直接删除，另起一个序号不一样的 Pod 时，无法通过内置 StatefulSet 实现。

增强型 StatefulSet 控制器 基于内置 StatefulSet 实现，新增了自由控制 Pods 序号的功能。本文介绍如何在 TiDB Operator 中使用。

11.1.3.1 开启

1. 载入 Advanced StatefulSet 的 CRD 文件：

    - Kubernetes 1.16 之前版本：

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/manifests/advanced-statefulset-crd.v1beta1.
    ↪ yaml
```

- Kubernetes 1.16 及之后版本:

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/manifests/advanced-statefulset-crd.v1.yaml
```

2. 在 TiDB Operator chart 的 values.yaml 中启用 AdvancedStatefulSet 特性:

```
features:
- AdvancedStatefulSet=true
advancedStatefulset:
  create: true
```

3. 升级 TiDB Operator，具体可参考升级 TiDB Operator 文档。

4. 升级 TiDB Operator 后，通过以下命令检查是否成功部署 AdvancedStatefulSet Controller:

```
kubectl get pods -n ${operator-ns} --selector app.kubernetes.io/
    ↪ component=advanced-statefulset-controller
```

点击查看期望输出

```
NAME                                             READY      STATUS
    ↪ RESTARTS  AGE
advanced-statefulset-controller-67885c5dd9-f522h 1/1    Running  0
    ↪          10s
```

> **注意:**
>
> TiDB Operator 通过开启 AdvancedStatefulSet 特性，会将当前 StatefulSet 对象转换成 AdvancedStatefulSet 对象。但是，TiDB Operator 不支持在关闭 AdvancedStatefulSet 特性后，自动从 AdvancedStatefulSet 转换为 Kubernetes 内置的 StatefulSet 对象。

11.1.3.2 使用

11.1.3.2.1 通过 kubectl 查看 AdvancedStatefulSet 对象

AdvancedStatefulSet 数据格式与 StatefulSet 完全一致，但以 CRD 方式实现，别名为 asts ，可通过以下方法查看命名空间下的对象。

```
kubectl get -n ${namespace} asts
```

11.1.3.2.2 操作 TidbCluster 对象指定 pod 进行缩容

使用增强型 StatefulSet 时，在对 TidbCluster 进行缩容时，除了减少副本数，可同时通过配置 annotations 指定对 PD，TiDB 或 TiKV 组件下任意一个 Pod 进行缩容。

比如：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: asts
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 4
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
    config: {}
```

上述配置会部署 4 个 TiKV 实例，分别为 basic-tikv-0，basic-tikv-1，...，basic-tikv-3。若想缩容掉 basic-tikv-1 需要修改 `spec.tikv.replicas` 为 3，同时配置以下 annotations:

```
metadata:
  annotations:
    tikv.tidb.pingcap.com/delete-slots: '[1]'
```

> 注意：
>
> 对 `replicas` 和 `delete-slots annotation` 的修改需在同一个操作中完成，不然控制器会根据修改一般的期望进行操作。

完整例子如下：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  annotations:
    tikv.tidb.pingcap.com/delete-slots: '[1]'
  name: asts
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
    config: {}
```

支持的 annotations 为：

- `pd.tidb.pingcap.com/delete-slots`：指定 PD 组件需要删除的 Pod 序号。
- `tidb.tidb.pingcap.com/delete-slots`：指定 TiDB 组件需要删除的 Pod 序号。
- `tikv.tidb.pingcap.com/delete-slots`：指定 TiKV 组件需要删除的 Pod 序号。

其中 Annotation 值为 JSON 的整数数组，比如 `[0]`, `[0,1]`, `[1,3]` 等。

11.1.3.2.3 操作 TidbCluster 对象在指定位置进行扩容

对前面缩容进行反向操作，即可恢复 basic-tikv-1。

> **注意：**
>
> 同常规 StatefulSet 缩容一样，并不会主动删除 Pod 关联的 PVC，若想避免使用之前数据，在原位置处扩容之前，需主动删除之前关联的 PVC。

例子如下：

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  annotations:
    tikv.tidb.pingcap.com/delete-slots: '[]'
  name: asts
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 4
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
    config: {}
```

其中 delete-slots annotations 可留空，也可完全删除。

11.1.4 TiDB Operator 准入控制器

Kubernetes 在 1.9 版本引入了 动态准入机制，从而使得拥有对 Kubernetes 中的各类资源进行修改与验证的功能。在 TiDB Operator 中，我们也同样使用了动态准入机制来帮助我们进行相关资源的修改、验证与运维。

11.1.4.1 先置条件

TiDB Operator 准入控制器与大部分 Kubernetes 平台上产品的准入控制器较为不同，TiDB Operator 通过扩展 API-Server 与 WebhookConfiguration 的两个机制组合而成。所以需要 Kubernetes 集群启用聚合层功能，通常情况下这个功能已经默认开启。如需查看是否开启聚合层功能，请参考启用 Kubernetes Apiserver 标志。

11.1.4.2 开启 TiDB Operator 准入控制器

TiDB Operator 在默认安装情况下不会开启准入控制器，你需要手动开启:

1. 修改 Operator 的 `values.yaml`

   开启 Operator Webhook 特性:

```
admissionWebhook:
  create: true
```

   默认情况下，如果你的 Kubernetes 集群版本大于等于 v1.13.0，你可以通过上述配置直接开启 Webhook 功能。

   如果你的 Kubernetes 集群版本小于 v1.13.0，你需要执行以下命令，将得到的返回值配置在 `values.yaml` 中的 `admissionWebhook.cabundle`:

```
kubectl get configmap -n kube-system extension-apiserver-authentication
    ↪  -o=jsonpath='{.data.client-ca-file}' | base64 | tr -d '\n'
```

```
admissionWebhook:
  # 将上述命令的返回值填写到 admissionWebhook.cabundle 中
  cabundle: <cabundle>
```

2. 配置失败策略

   在 Kubernetes 1.15 版本之前，动态准入机制的管理机制的粒度较粗并且并不方便去使用。所以为了防止 TiDB Operator 的动态准入机制影响全局集群，我们需要配置失败策略。

   对于 Kubernetes 1.15 以下的版本，我们推荐将 TiDB Operator 失败策略配置为 `Ignore`，从而防止 TiDB Operator 的 admission webhook 出现异常时影响整个集群。

```
......
failurePolicy:
    validation: Ignore
    mutation: Ignore
```

   对于 Kubernetes 1.15 及以上的版本，我们推荐给 TiDB Operator 失败策略配置为 `Failure`，由于 Kubernetes 1.15 及以上的版本中，动态准入机制已经有了基于 Label 的筛选机制，所以不会由于 TiDB Operator 的 admission webhook 出现异常而影响整个集群。

```
......
failurePolicy:
    validation: Fail
    mutation: Fail
```

3. 安装/更新 TiDB Operator

   修改完 values.yaml 文件中的上述配置项以后，进行 TiDB Operator 部署或者更新。安装与更新 TiDB Operator 请参考在 Kubernetes 上部署 TiDB Operator。

11.1.4.3 为 TiDB Operator 准入控制器设置 TLS 证书

在默认情况下，TiDB Operator 准入控制器与 Kubernetes api-server 之间跳过了 TLS 验证环节，你可以通过以下步骤手动开启并配置 TiDB Operator 准入控制器与 Kubernetes api-server 之间的 TLS 验证。

1. 生成自定义证书

   参考使用 cfssl 系统颁发证书的第一步至第四步，生成自定义 CA 文件。对于 ca-config.json，我们使用如下配置:

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

   当执行至第四步以后，通过 ls 命令执行，cfssl 文件夹下应该有以下文件:

```
ca-config.json  ca-csr.json  ca-key.pem  ca.csr  ca.pem
```

2. 生成 TiDB Operator 准入控制器证书

   首先生成默认的 webhook-server.json 文件:

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

然后将 webhook-server.json 文件的内容修改如下:

```
{
    "CN": "TiDB Operator Webhook",
    "hosts": [
        "tidb-admission-webhook.<namespace>",
        "tidb-admission-webhook.<namespace>.svc",
        "tidb-admission-webhook.<namespace>.svc.cluster",
        "tidb-admission-webhook.<namespace>.svc.cluster.local"
    ],
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "US",
            "L": "CA",
            "O": "PingCAP",
            "ST": "Beijing",
            "OU": "TiDB"
        }
    ]
}
```

其中 <namespace> 为 TiDB Operator 部署的命名空间。

然后生成 TiDB Operator Webhook Server 端证书:

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

执行完上述命令后，通过 ls | grep webhook-server 命令应该能查询到以下文件:

```
webhook-server-key.pem
webhook-server.csr
webhook-server.json
webhook-server.pem
```

3. 在 Kubernetes 集群中创建 Secret

```
kubectl create secret generic <secret-name> --namespace=<namespace> --
    ↪ from-file=tls.crt=~/cfssl/webhook-server.pem --from-file=tls.key
    ↪ =~/cfssl/webhook-server-key.pem --from-file=ca.crt=~/cfssl/ca.pem
```

4. 修改 values.yaml 并安装或升级 TiDB Operator

   获取 ca.crt 的值：

```
kubectl get secret <secret-name> --namespace=<release-namespace> -o=
    ↪ jsonpath='{.data.ca\.crt}'
```

   将 values.yaml 中下述配置按说明来进行配置：

```
admissionWebhook:
  apiservice:
    insecureSkipTLSVerify: false # 开启 TLS 验证
    tlsSecret: "<secret-name>" # 将上文中所创建的 secret 的 name
        ↪ 填写在这里
    caBundle: "<caBundle>" # 将上文中 ca.crt 的值填入此处
```

   修改完 values.yaml 文件中上述配置项以后进行 TiDB Operator 部署或者更新。安装 TiDB Operator 请参考在 Kubernetes 上部署 TiDB Operator，升级 TiDB Operator 请参考升级 TiDB Operator

11.1.4.4 TiDB Operator 准入控制器功能

TiDB Operator 通过准入控制器的帮助实现了许多功能。我们将在这里介绍各个资源的准入控制器与其相对应的功能。

- Pod 验证准入控制器：

  Pod 准入控制器提供了对 PD/TiKV/TiDB 组件安全下线与安全上线的保证，通过 Pod 验证准入控制器，我们可以实现重启 Kubernetes 上的 TiDB 集群。该组件在准入控制器开启的情况下默认开启。

```
admissionWebhook:
  validation:
    pods: true
```

- StatefulSet 验证准入控制器：

  StatefulSet 验证准入控制器帮助实现 TiDB 集群中 TiDB/TiKV 组件的灰度发布，该组件在准入控制器开启的情况下默认关闭。

```
admissionWebhook:
  validation:
    statefulSets: false
```

  通过 tidb.pingcap.com/tikv-partition 和 tidb.pingcap.com/tidb-partition 这两个 annotation，你可以控制 TiDB 集群中 TiDB 与 TiKV 组件的灰度发布。你可以通过以下方式对 TiDB 集群的 TiKV 组件设置灰度发布，其中 partition=2 的效果等同于 StatefulSet 分区

```
$ kubectl annotate tidbcluster ${name} -n ${namespace} tidb.pingcap.
    ↪ com/tikv-partition=2
tidbcluster.pingcap.com/${name} annotated
```

执行以下命令取消灰度发布设置：

```
$ kubectl annotate tidbcluster ${name} -n ${namespace} tidb.pingcap.
    ↪ com/tikv-partition-
tidbcluster.pingcap.com/${name} annotated
```

以上设置同样适用于 TiDB 组件。

- TiDB Operator 资源验证准入控制器:

  TiDB Operator 资源验证准入控制器帮助实现针对 `TidbCluster`、`TidbMonitor` 等 TiDB Operator 自定义资源的验证，该组件在准入控制器开启的情况下默认关闭。

```
admissionWebhook:
  validation:
    pingcapResources: false
```

  举个例子，对于 `TidbCluster` 资源，TiDB Operator 资源验证准入控制器将会检查其 `spec` 字段中的必要字段。如果在 `TidbCluster` 创建或者更新时发现检查不通过，比如同时没有定义 `spec.pd.image` 或者 `spec.pd.baseImage` 字段，TiDB Operator 资源验证准入控制器将会拒绝这个请求。

- Pod 修改准入控制器:

  Pod 修改准入控制器帮助我们在弹性伸缩场景下实现 TiKV 的热点调度功能，在启用 TidbCluster 弹性伸缩中需要开启该控制器。该组件在准入控制器开启的情况下默认开启。

```
admissionWebhook:
  mutation:
    pods: true
```

- TiDB Operator 资源修改准入控制器:

  TiDB Operator 资源修改准入控制器帮助我们实现 TiDB Operator 相关自定义资源的默认值填充工作，如 `TidbCluster`，`TidbMonitor` 等。该组件在准入控制器开启的情况下默认开启。

```
admissionWebhook:
  mutation:
    pingcapResources: true
```

11.2 TiDB in Kubernetes Sysbench 性能测试

随着 TiDB Operator GA 发布，越来越多用户开始使用 TiDB Operator 在 Kubernetes 中部署管理 TiDB 集群。在本次测试中，我们选择 GKE 平台做了一次深入、全方位的测试，方便大家了解 TiDB 在 Kubernetes 中性能影响因素。

11.2.1 目的

- 测试典型公有云平台上 TiDB 性能数据
- 测试公有云平台磁盘、网络、CPU 以及不同 Pod 网络下对 TiDB 性能的影响

11.2.2 环境

11.2.2.1 版本与配置

本次测试统一使用 TiDB v3.0.1 版本进行测试。

TiDB Operator 使用 v1.0.0 版本。

PD、TiDB 和 TiKV 实例数均为 3 个。各组件分别作了如下配置，未配置部分使用默认值。

PD:

```
[log]
level = "info"
[replication]
location-labels = ["region", "zone", "rack", "host"]
```

TiDB:

```
[log]
level = "error"
[prepared-plan-cache]
enabled = true
[tikv-client]
max-batch-wait-time = 2000000
```

TiKV:

```
log-level = "error"
[server]
status-addr = "0.0.0.0:20180"
grpc-concurrency = 6
[readpool.storage]
normal-concurrency = 10
[rocksdb.defaultcf]
block-cache-size = "14GB"
```

```
[rocksdb.writecf]
block-cache-size = "8GB"
[rocksdb.lockcf]
block-cache-size = "1GB"
[raftstore]
apply-pool-size = 3
store-pool-size = 3
```

11.2.2.2 TiDB 数据库参数配置

```
set global tidb_hashagg_final_concurrency=1;
set global tidb_hashagg_partial_concurrency=1;
set global tidb_disable_txn_auto_retry=0;
```

11.2.2.3 硬件配置

11.2.2.3.1 机器型号

在单可用区测试中，我们选择了如下型号机器：

组件	实例类型	数量
PD	n1-standard-4	3
TiKV	c2-standard-16	3
TiDB	c2-standard-16	3
Sysbench	c2-standard-30	1

分别在多可用区和单可用区中对 TiDB 进行性能测试，并将结果相对比。在测试时 (2019.08)，一个 GCP 区域 (region) 下不存在三个能同时提供 c2 机器的可用区，所以我们选择了如下机器型号进行测试：

组件	实例类型	数量
PD	n1-standard-4	3
TiKV	n1-standard-16	3
TiDB	n1-standard-16	3
Sysbench	n1-standard-16	3

在高并发读测试中，压测端 sysbench 对 CPU 需求较高，需选择多核较高配置型号，避免压测端成为瓶颈。

注意：

> GCP 不同的区域可用机型不同。同时测试中发现磁盘性能表现也存在差异，我们统一在 us-central1 中申请机器测试。

11.2.2.3.2 磁盘

GKE 当前的 NVMe 磁盘还在 Alpha 阶段，使用需特殊申请，不具备普遍意义。测试中，本地 SSD 盘统一使用 iSCSI 接口类型。挂载参数参考官方建议增加了 `discard,nobarrier` 选项。完整挂载例子如下：

```
sudo mount -o defaults,nodelalloc,noatime,discard,nobarrier /dev/[
    ↪ LOCAL_SSD_ID] /mnt/disks/[MNT_DIR]
```

11.2.2.3.3 网络

GKE 网络模式使用具备更好扩展性以及功能强大的 VPC-Native 模式。在性能对比中，我们分别对 TiDB 使用 Kubernetes Pod 和 Host 网络分别做了测试。

11.2.2.3.4 CPU

在单可用集群测试中，我们为 TiDB/TiKV 选择 c2-standard-16 机型测试。在单可用与多可用集群的对比测试中，因为 GCP 区域 (region) 下不存在三个能同时申请 c2-standard-16 机型的可用区，所以我们选择了 n1-standard-16 机型测试。

11.2.2.4 操作系统及参数

GKE 支持两种操作系统：COS (Container Optimized OS) 和 Ubuntu 。Point Select 测试在两种操作系统中进行，并将结果进行了对比。其余测试中，统一使用 Ubuntu 系统进行测试。

内核统一做了如下配置:

```
sysctl net.core.somaxconn=32768
sysctl vm.swappiness=0
sysctl net.ipv4.tcp_syncookies=0
```

同时对最大文件数配置为 1000000。

11.2.2.5 Sysbench 版本与运行参数

本次测试中 sysbench 版本为 1.0.17。并在测试前统一使用 `oltp_common` 的 `prewarm` 命令进行数据预热。

11.2.2.5.1 初始化

```
sysbench \
  --mysql-host=${tidb_host} \
  --mysql-port=4000 \
  --mysql-user=root \
  --mysql-db=sbtest \
  --time=600 \
  --threads=16 \
  --report-interval=10 \
  --db-driver=mysql \
  --rand-type=uniform \
  --rand-seed=$RANDOM \
  --tables=16 \
  --table-size=10000000 \
  oltp_common \
  prepare
```

`${tidb_host}` 为 TiDB 的数据库地址，根据不同测试需求选择不同的地址，比如 Pod IP、Service 域名、Host IP 以及 Load Balancer IP（下同）。

11.2.2.5.2 预热

```
sysbench \
  --mysql-host=${tidb_host} \
  --mysql-port=4000 \
  --mysql-user=root \
  --mysql-db=sbtest \
  --time=600 \
  --threads=16 \
  --report-interval=10 \
  --db-driver=mysql \
  --rand-type=uniform \
  --rand-seed=$RANDOM \
  --tables=16 \
  --table-size=10000000 \
  oltp_common \
  prewarm
```

11.2.2.5.3 压测

```
sysbench \
  --mysql-host=${tidb_host} \
  --mysql-port=4000 \
  --mysql-user=root \
```

```
--mysql-db=sbtest \
--time=600 \
--threads=${threads} \
--report-interval=10 \
--db-driver=mysql \
--rand-type=uniform \
--rand-seed=$RANDOM \
--tables=16 \
--table-size=10000000 \
${test} \
run
```

${test} 为 sysbench 的测试 case。我们选择了 oltp_point_select、oltp_update_index、oltp_update_no_index、oltp_read_write 这几种。

11.2.3 测试报告

11.2.3.1 单可用区测试

11.2.3.1.1 Pod Network vs Host Network

Kubernetes 允许 Pod 运行在 Host 网络模式下。此部署方式适用于 TiDB 实例独占机器且没有端口冲突的情况。我们分别在两种网络模式下做了 Point Select 测试。

此次测试中，操作系统为 COS。

Pod Network:

Threads	QPS	95% latency(ms)
150	246386.44	0.95
300	346557.39	1.55
600	396715.66	2.86
900	407437.96	4.18
1200	415138.00	5.47
1500	419034.43	6.91

Host Network:

Threads	QPS	95% latency(ms)
150	255981.11	1.06
300	366482.22	1.50
600	421279.84	2.71
900	438730.81	3.96
1200	441084.13	5.28
1500	447659.15	6.67

QPS 对比：

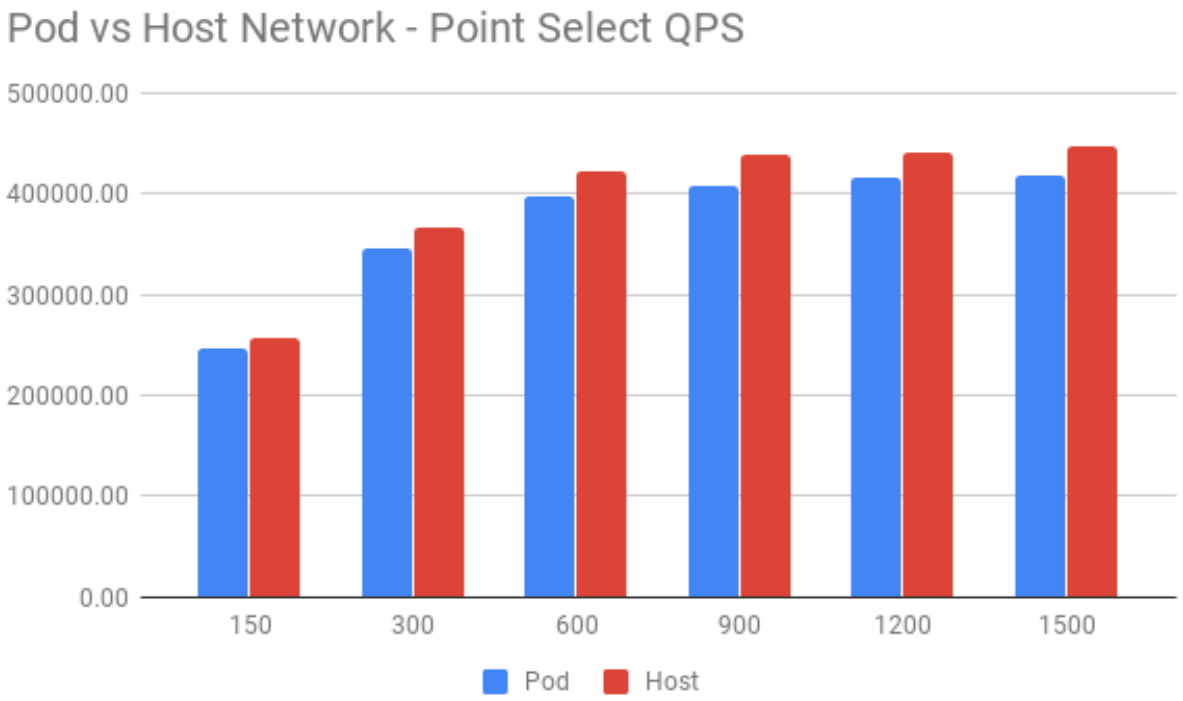


Figure 4: Pod vs Host Network

Latency 对比：

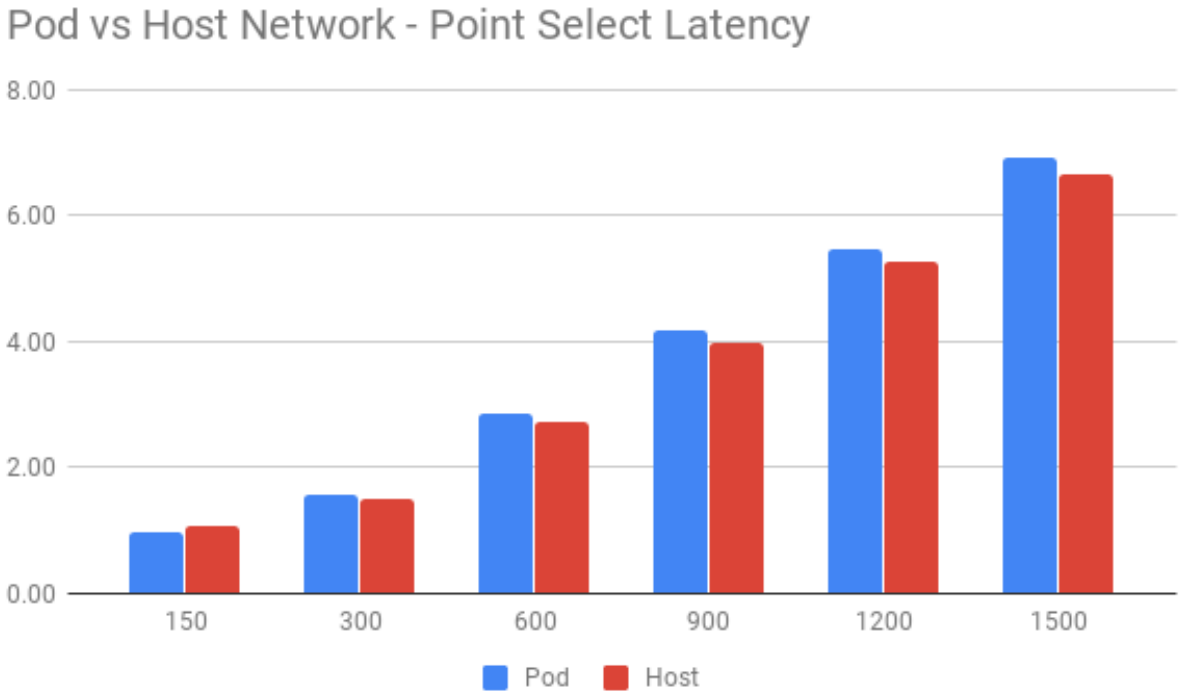


Figure 5: Pod vs Host Network

从图中可以看到 Host 网络下整体表现略好于 Pod 网络。

11.2.3.1.2 Ubuntu vs COS

GKE 平台可以为节点选择 Ubuntu 和 COS 两种操作系统。本次测试中，分别在两种操作系统中进行了 Point Select 测试。

此次测试中 Pod 网络模式为 Host。

COS:

Threads	QPS	95% latency(ms)
150	255981.11	1.06
300	366482.22	1.50
600	421279.84	2.71
900	438730.81	3.96
1200	441084.13	5.28
1500	447659.15	6.67

Ubuntu:

Threads	QPS	95% latency(ms)
150	290690.51	0.74
300	422941.17	1.10
600	476663.44	2.14
900	484405.99	3.25
1200	489220.93	4.33
1500	489988.97	5.47

QPS **对比：**

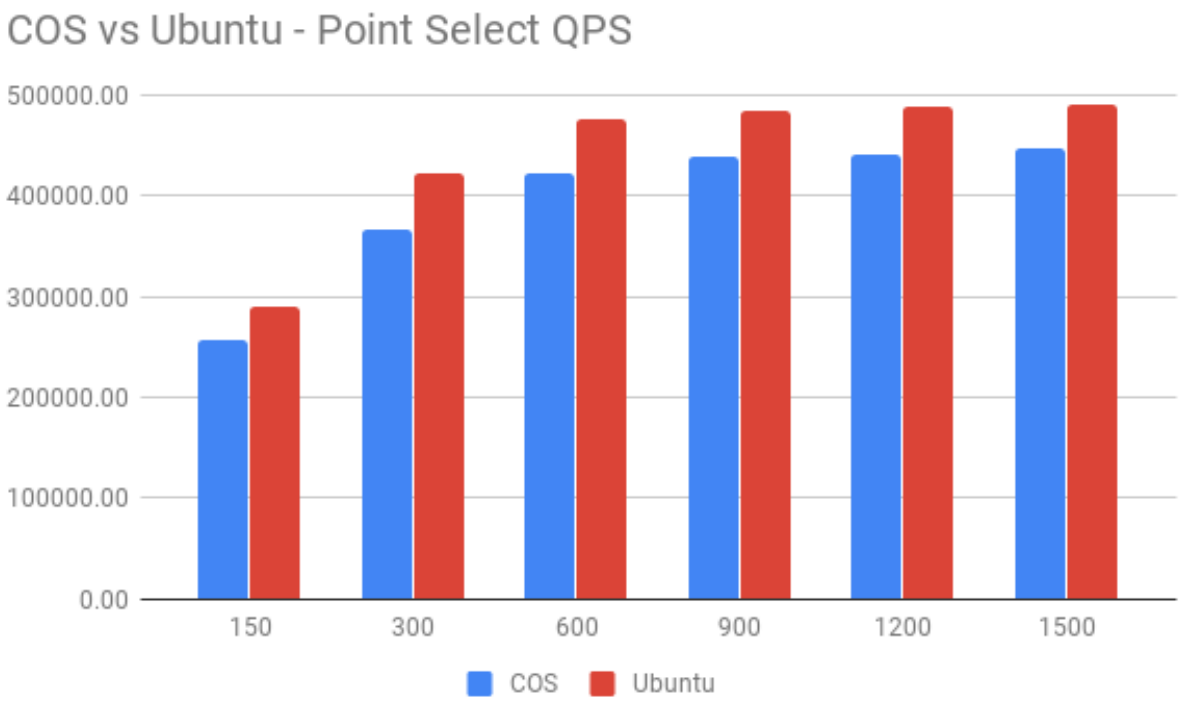


Figure 6: COS vs Ubuntu

Latency **对比：**

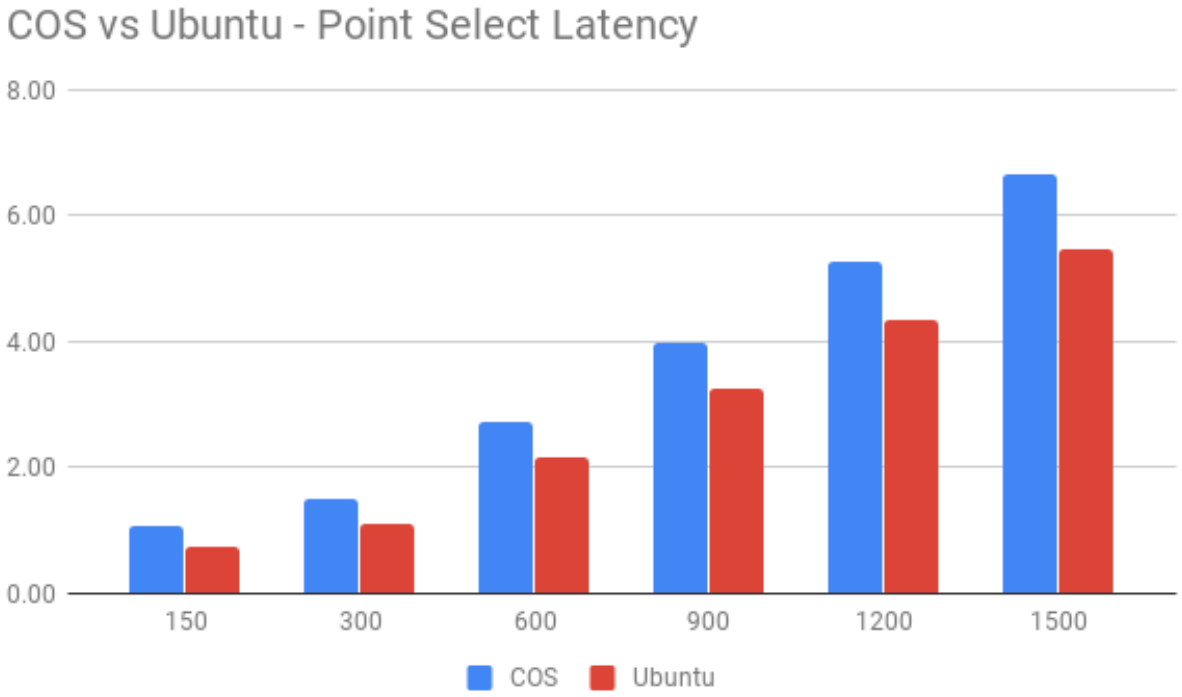


Figure 7: COS vs Ubuntu

从图中可以看到 Host 模式下，在单纯的 Point Select 测试中，TiDB 在 Ubuntu 系统中的表现比在 COS 系统中的表现要好。

> **注意：**
>
> 此测试只针对单一测试集进行了测试，表明不同的操作系统、不同的优化与默认设置，都有可能影响性能，所以我们在此不对操作系统做推荐。COS 系统专为容器优化，在安全性、磁盘性能做了优化，在 GKE 是官方推荐系统。

11.2.3.1.3 K8S Service vs GCP LoadBalancer

通过 Kubernetes 部署 TiDB 集群后，有两种访问 TiDB 集群的场景：集群内通过 Service 访问或集群外通过 Load Balancer IP 访问。本次测试中分别对这两种情况进行了对比测试。

此次测试中操作系统为 Ubuntu，Pod 为 Host 网络。

Service:

Threads	QPS	95% latency(ms)
150	290690.51	0.74
300	422941.17	1.10
600	476663.44	2.14
900	484405.99	3.25
1200	489220.93	4.33
1500	489988.97	5.47

Load Balancer:

Threads	QPS	95% latency(ms)
150	255981.11	1.06
300	366482.22	1.50
600	421279.84	2.71
900	438730.81	3.96
1200	441084.13	5.28
1500	447659.15	6.67

QPS 对比：

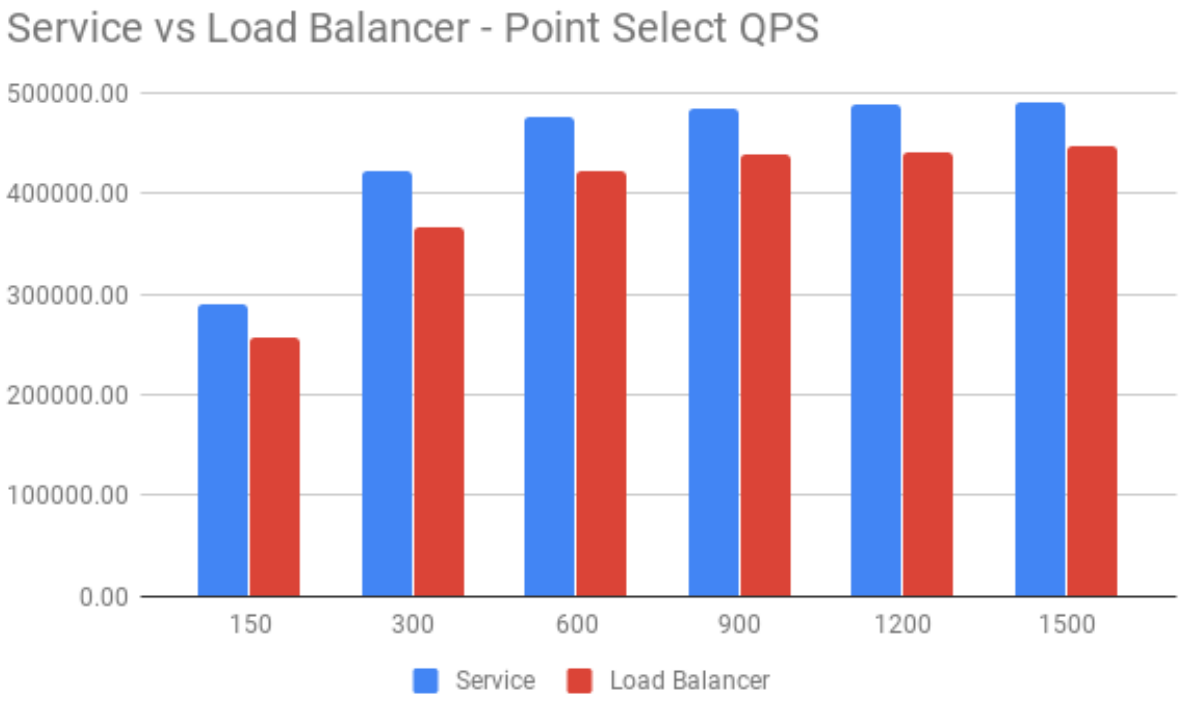


Figure 8: Service vs Load Balancer

Latency 对比：

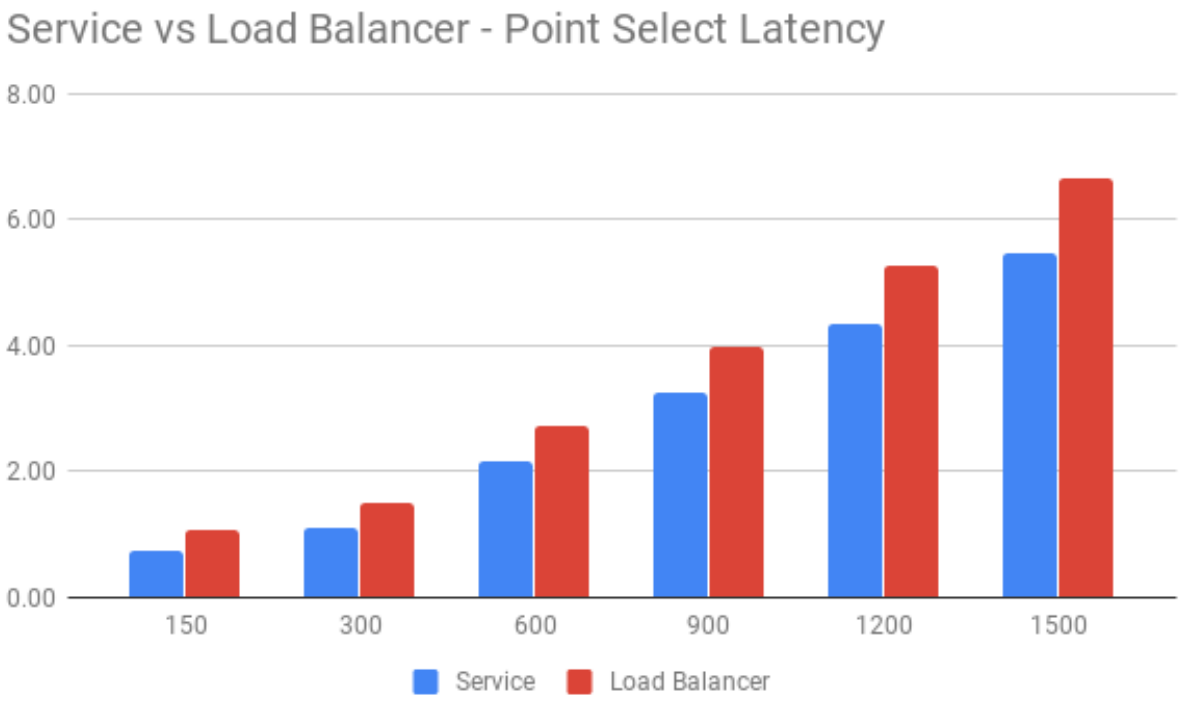


Figure 9: Service vs Load Balancer

从图中可以看到在单纯的 Point Select 测试中，使用 Kubernetes Service 访问 TiDB 时的表现比使用 GCP Load Balancer 访问时要好。

11.2.3.1.4 n1-standard-16 vs c2-standard-16

在 Point Select 读测试中，TiDB 的 CPU 占用首先达到 1400% (16 cores) 以上，此时 TiKV CPU 占用约 1000% (16 cores) 。我们对比了普通型和计算优化型机器下 TiDB 的不同表现。其中 n1-stadnard-16 主频约 2.3G，c2-standard-16 主频约 3.1G。

此次测试中操作系统为 Ubuntu，Pod 为 Host 网络，使用 Service 访问 TiDB。

n1-standard-16:

Threads	QPS	95% latency(ms)
150	203879.49	1.37
300	272175.71	2.3
600	287805.13	4.1
900	295871.31	6.21
1200	294765.83	8.43
1500	298619.31	10.27

c2-standard-16:

Threads	QPS	95% latency(ms)
150	290690.51	0.74
300	422941.17	1.10
600	476663.44	2.14
900	484405.99	3.25
1200	489220.93	4.33
1500	489988.97	5.47

QPS 对比：

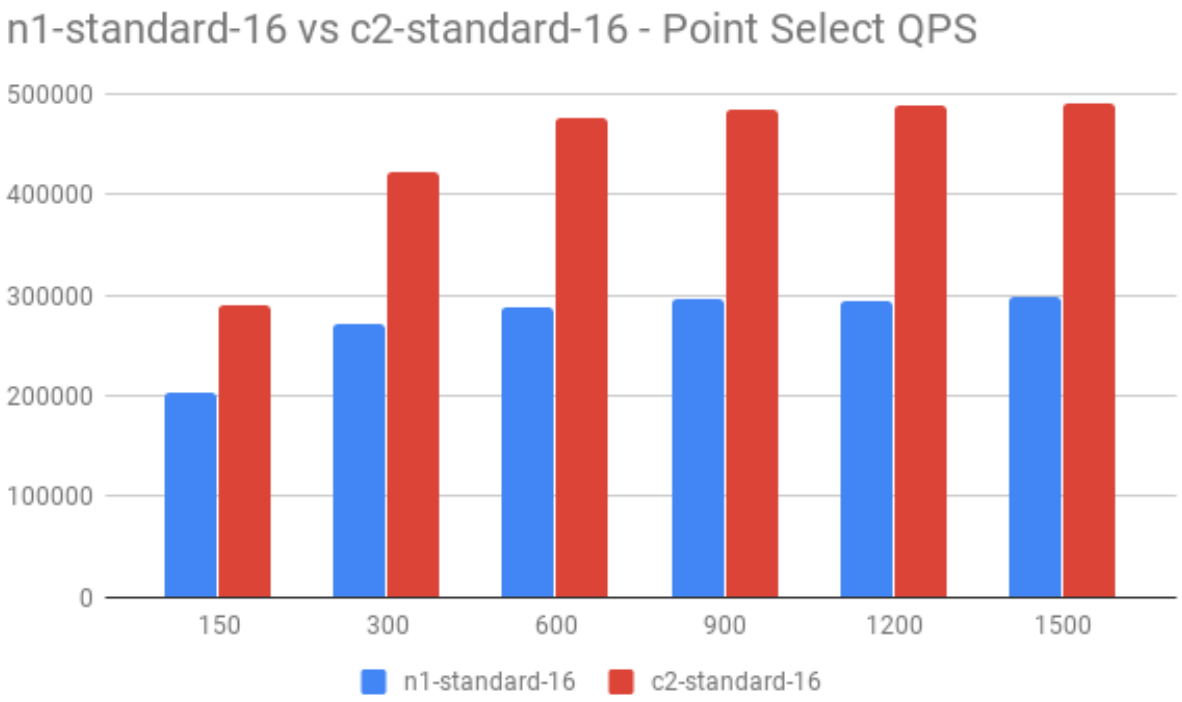


Figure 10: n1-standard-16 vs c2-standard-16

Latency 对比：

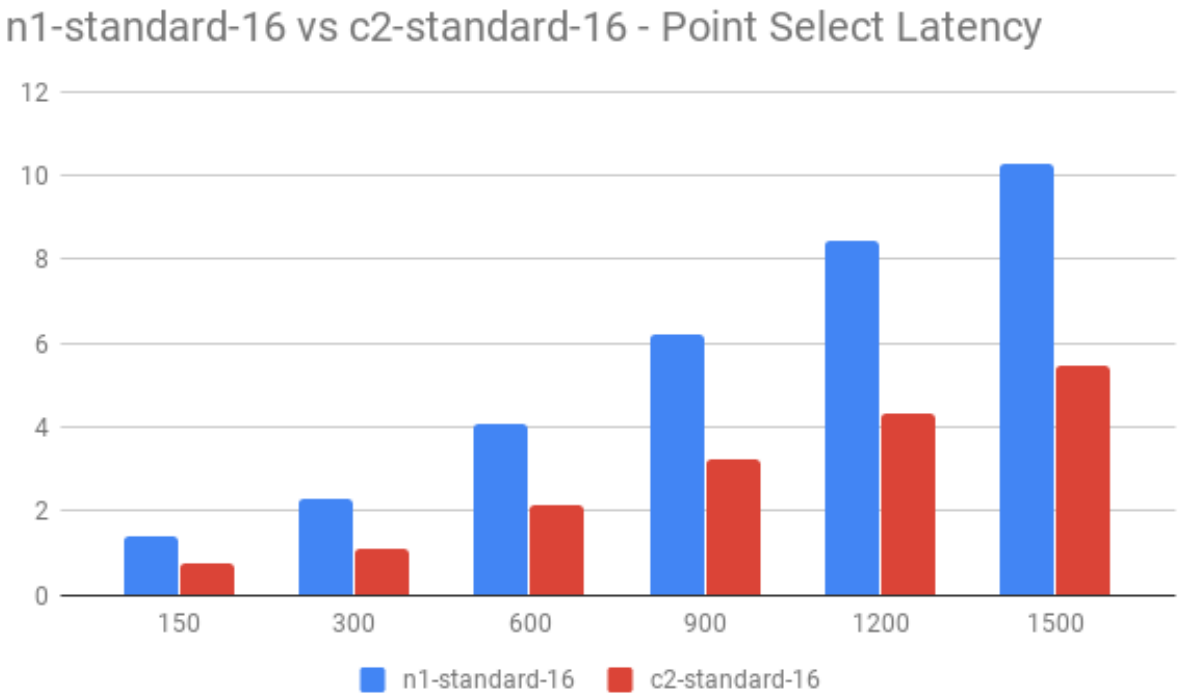


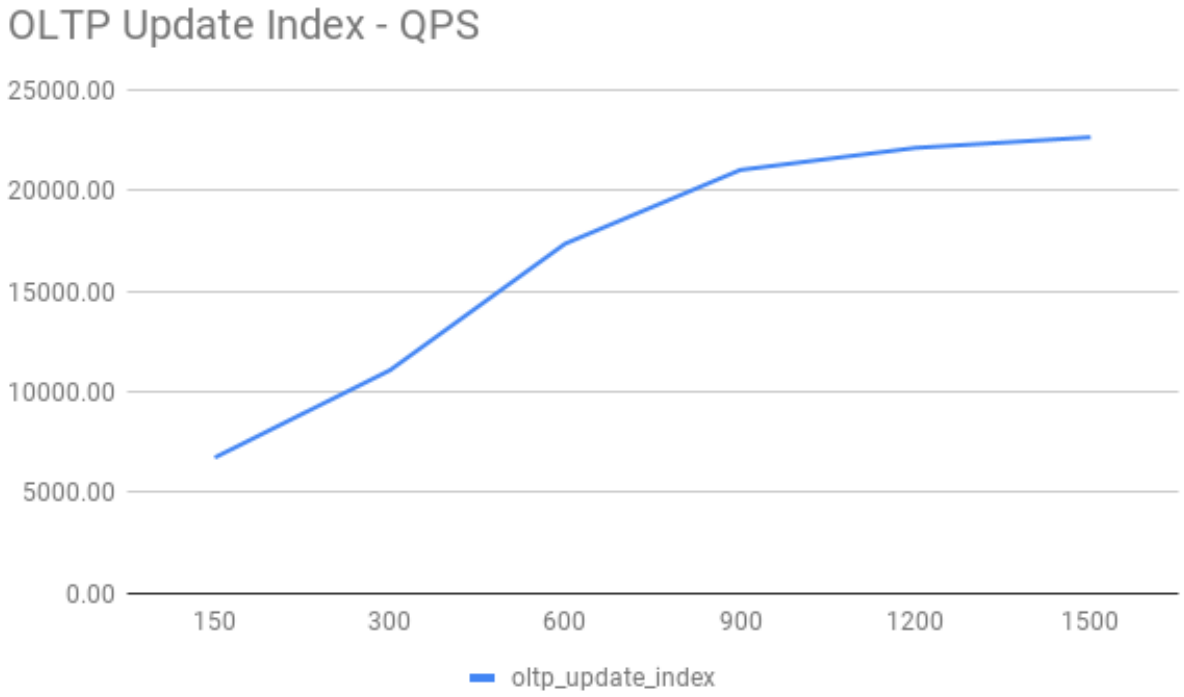
Figure 11: n1-standard-16 vs c2-standard-16

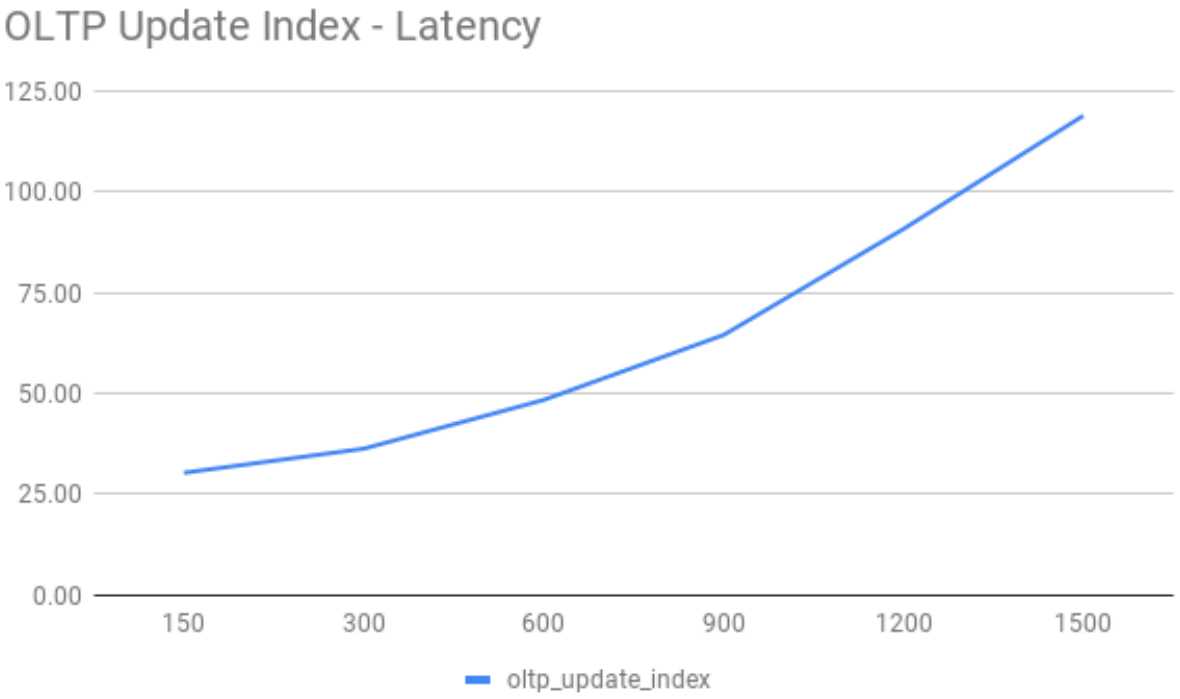
11.2.3.2 OLTP 其他测试

使用 Point Select 测试针对不同操作系统、不同网络情况做了对比测试后，也进行了 OLTP 测试集中的其他测试。这些测试统一使用 Ubuntu 系统、Host 模式并在集群使用 Service 访问 TiDB 集群。

11.2.3.2.1 OLTP Update Index

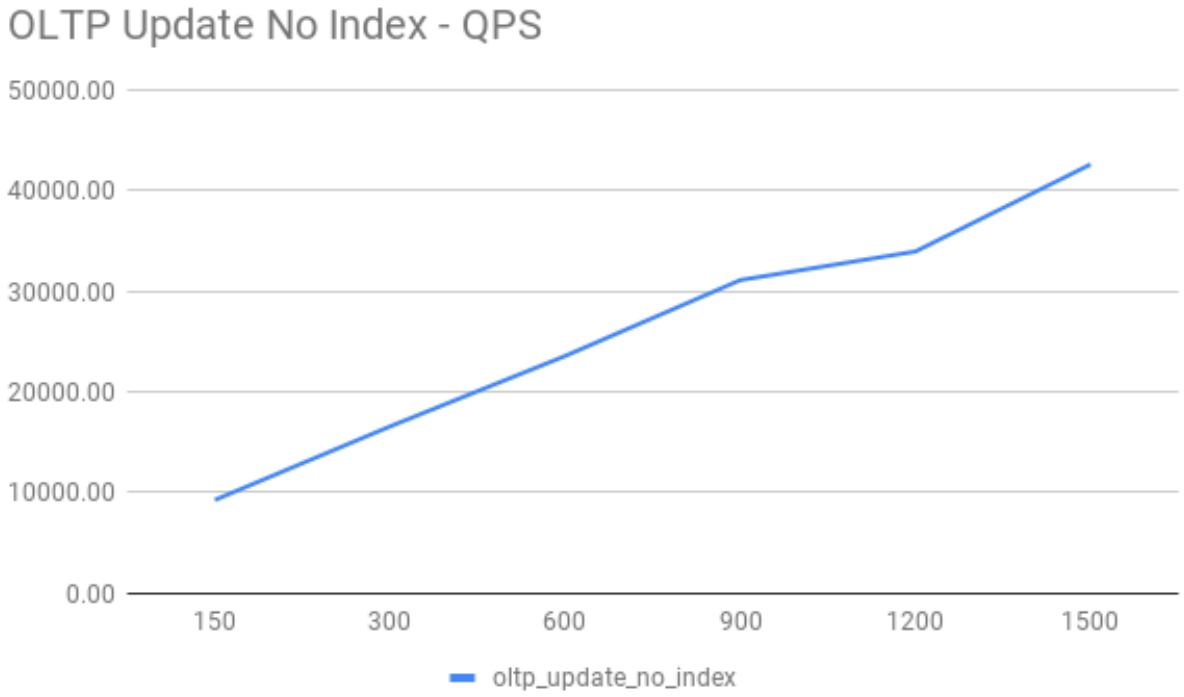
Threads	QPS	95% latency(ms)
150	6726.59	30.26
300	11067.55	36.24
600	17358.46	48.34
900	21025.23	64.47
1200	22121.87	90.78
1500	22650.13	118.92

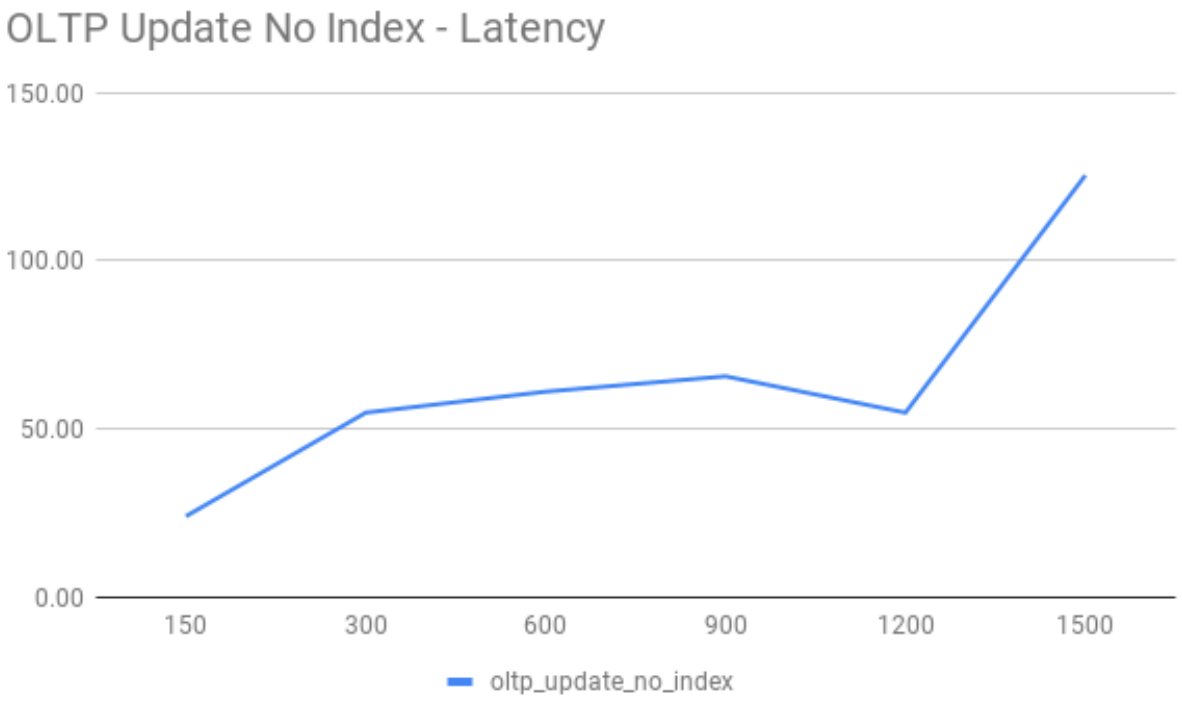




11.2.3.2.2 OLTP Update Non Index

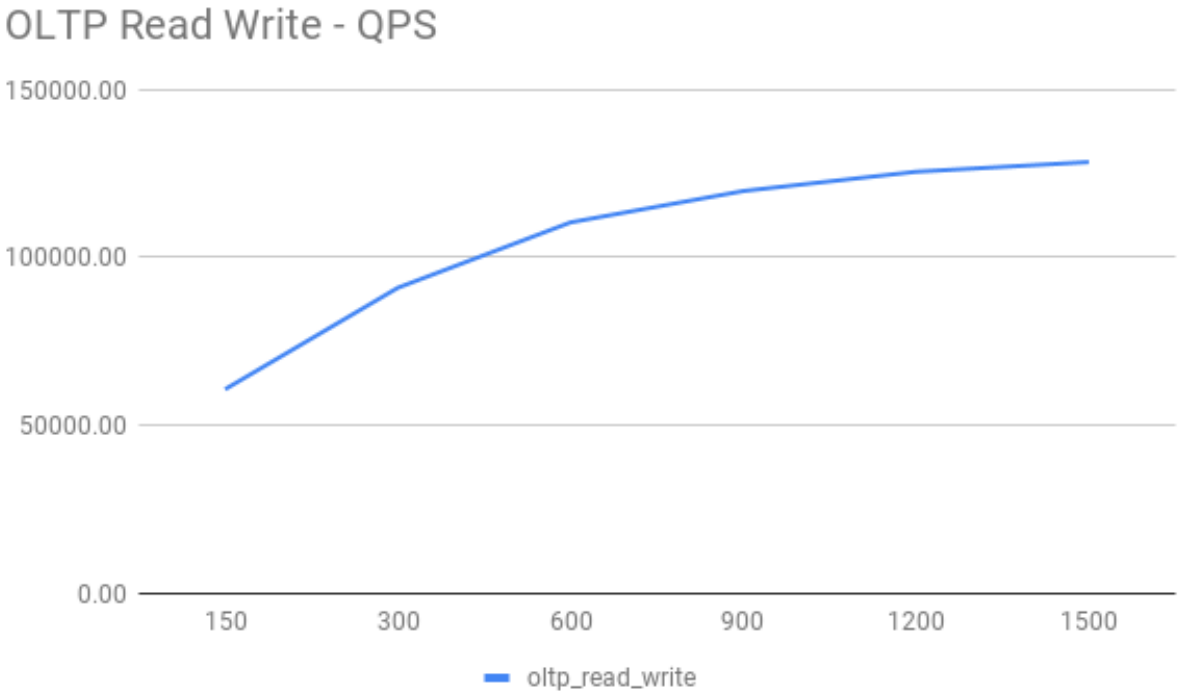
Threads	QPS	95% latency(ms)
150	9230.60	23.95
300	16543.63	54.83
600	23551.01	61.08
900	31100.10	65.65
1200	33942.60	54.83
1500	42603.13	125.52

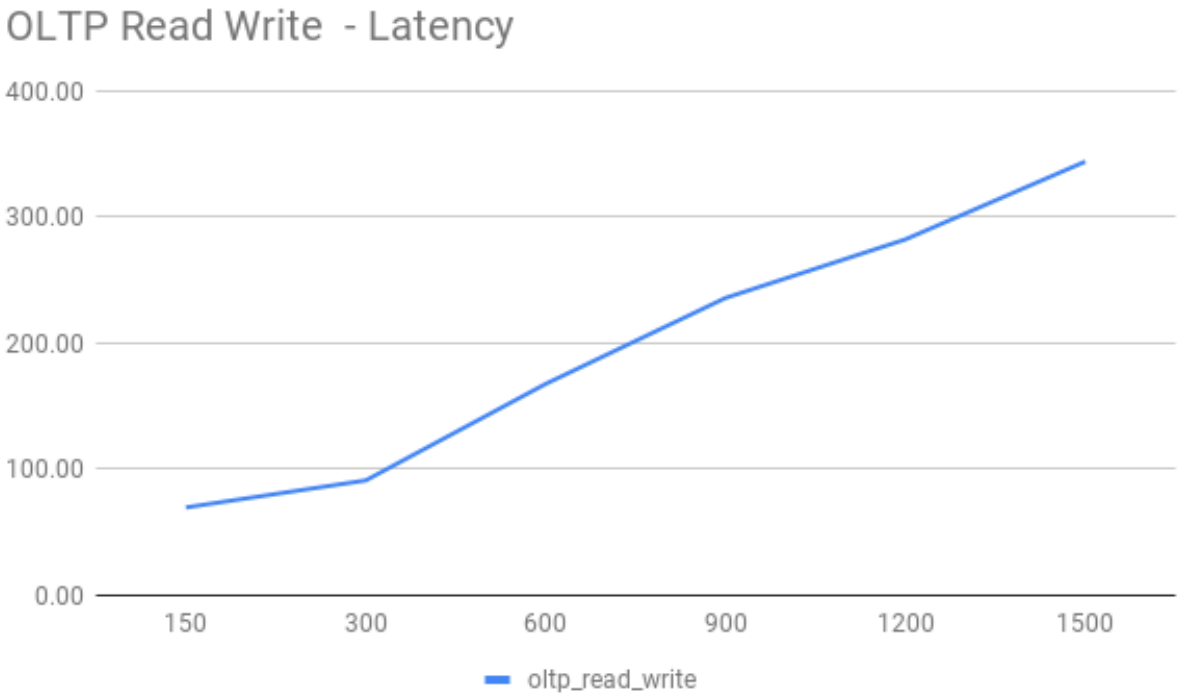




11.2.3.2.3 OLTP Read Write

Threads	QPS	95% latency(ms)
150	60732.84	69.29
300	91005.98	90.78
600	110517.67	167.44
900	119866.38	235.74
1200	125615.89	282.25
1500	128501.34	344.082





11.2.3.3 单可用区与多可用区对比

GCP 多可用区涉及跨 Zone 通信，网络延迟相比同 Zone 会少许增加。我们使用同样机器配置，对两种部署方案进行同一标准下的性能测试，了解多可用区延迟增加带来的影响。

单可用区：

Threads	QPS	95% latency(ms)
150	203879.49	1.37
300	272175.71	2.30
600	287805.13	4.10
900	295871.31	6.21
1200	294765.83	8.43
1500	298619.31	10.27

多可用区：

Threads	QPS	95% latency(ms)
150	141027.10	1.93
300	220205.85	2.91
600	250464.34	5.47
900	257717.41	7.70
1200	258835.24	10.09
1500	280114.00	12.75

QPS **对比：**

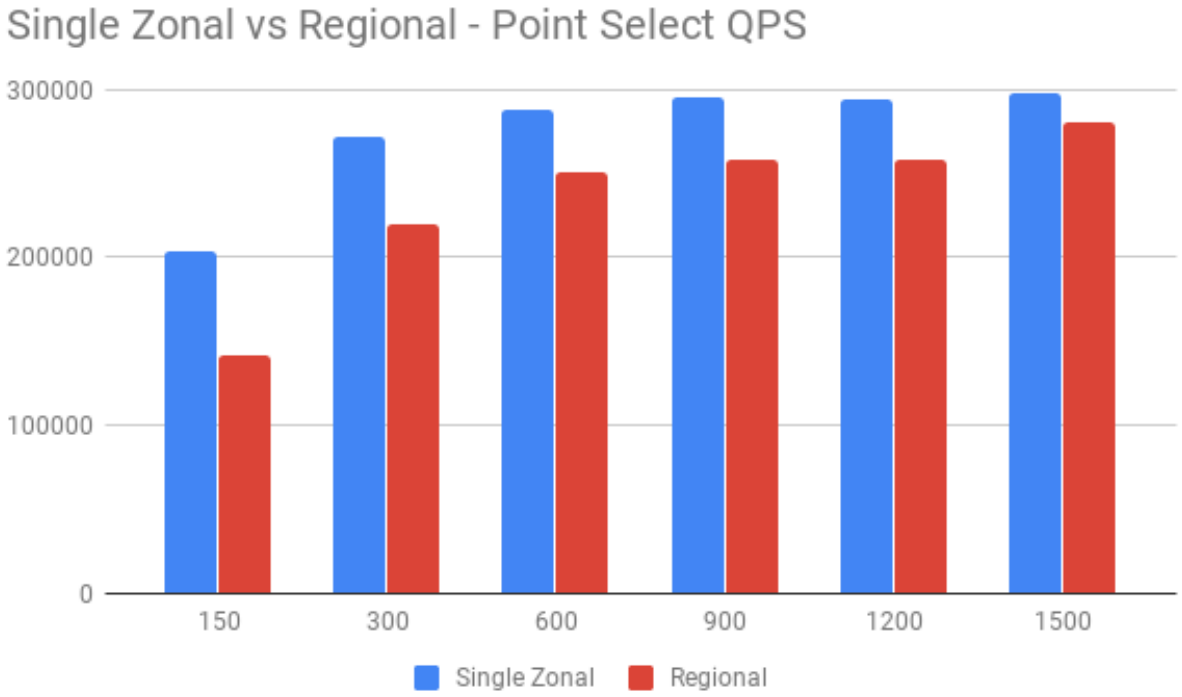


Figure 12: Single Zonal vs Regional

Latency 对比：

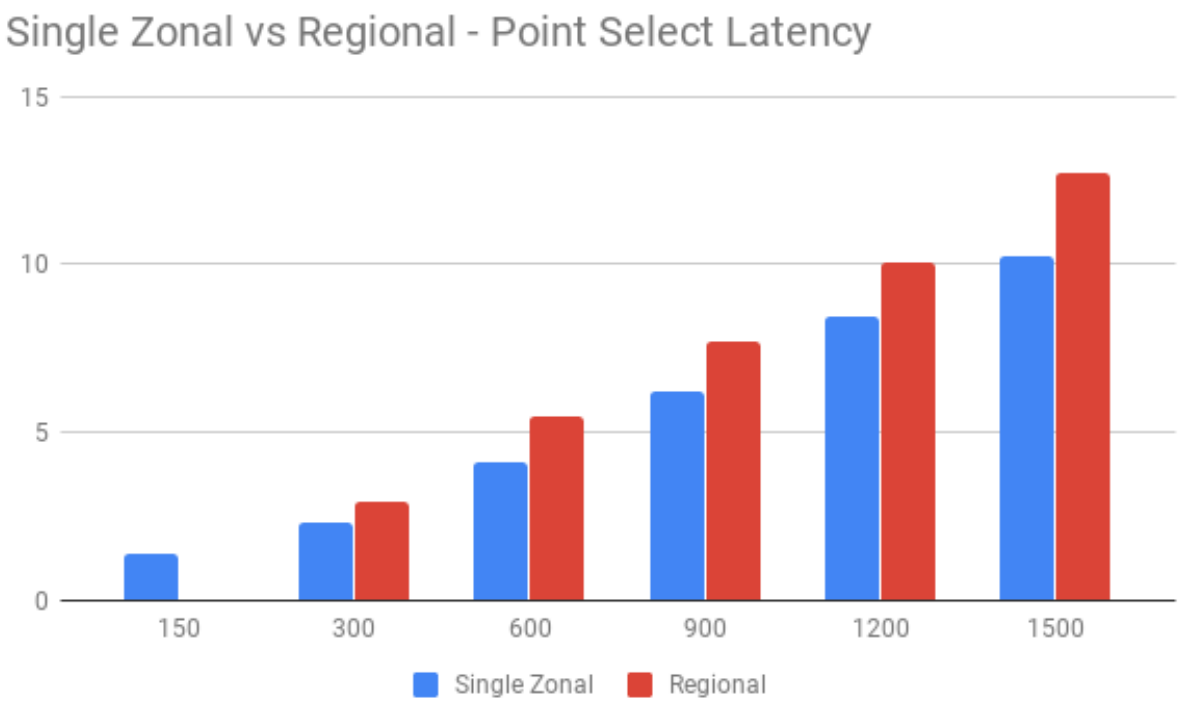


Figure 13: Single Zonal vs Regional

从图中可以看到并发压力增大后，网络额外延迟产生的影响越来越小，额外的网络延迟将不再是主要的性能瓶颈。

11.2.4 结语

此次测试主要将典型公有云部署 Kubernetes 运行 TiDB 集群的几种场景使用 sysbench 做了测试，了解不同因素可能带来的影响。从整体看，主要有以下几点：

- VPC-Native 模式下 Host 网络性能略好于 Pod 网络（~7%，以 QPS 差异估算，下同）
- GCP 的 Ubuntu 系统 Host 网络下单纯的读测试中性能略好于 COS (~9%)
- 使用 Load Balancer 在集群外访问，会略损失性能 (~5%)
- 多可用区下节点之间延迟增加，会对 TiDB 性能产生一定的影响（30% ~ 6%，随并发数增加而下降）
- Point Select 读测试主要消耗 CPU ，计算型机型相对普通型机器带来了很大 QPS 提升 (50% ~ 60%)

但要注意的是，这些因素可能随着时间变化，不同公有云下的表现可能会略有不同。在未来，我们将带来更多维度的测试。同时，sysbench 测试用例并不能完全代表实际业务场景，在做选择前建议模拟实际业务测试，并综合不同选择成本进行选择（机器成本、操作系统差异、Host 网络的限制等）。

11.3 API 参考文档

11.4 管理 TiDB 集群的 Command Cheat Sheet

本文提供管理 TiDB 集群的 Command Cheat Sheet。

11.4.1 kubectl

11.4.1.1 查看资源

- 查看 CRD：

```
kubectl get crd
```

- 查看 TidbCluster：

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

- 查看 TidbMonitor：

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

- 查看 Backup：

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

- 查看 BackupSchedule：

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

- 查看 Restore：

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

- 查看 TidbClusterAutoScaler：

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

- 查看 TidbInitializer：

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

- 查看 Advanced StatefulSet：

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

- 查看 Pod：

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

查看 TiKV Pod：

```
kubectl -n ${namespace} get pod -l app.kubernetes.io/component=tikv
```

持续观察 Pod 状态变化：

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

查看 Pod 详细信息：

```
kubectl -n ${namespace} describe pod ${name}
```

- 查看 Pod 所在 Node：

```
kubectl -n ${namespace} get pods -l "app.kubernetes.io/component=tidb,
    ↪ app.kubernetes.io/instance=${cluster_name}" -ojsonpath="{range .
    ↪ items[*]}{.spec.nodeName}{'\n'}{end}"
```

- 查看 Service：

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

- 查看 ConfigMap：

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

- 查看 PV：

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

查看集群使用的 PV:

```
kubectl get pv -l app.kubernetes.io/namespace=${namespace},app.
    ↪ kubernetes.io/managed-by=tidb-operator,app.kubernetes.io/instance
    ↪ =${cluster_name}
```

- 查看 PVC：

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

- 查看 StorageClass：

```
kubectl -n ${namespace} get sc
```

- 查看 StatefulSet：

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

查看 StatefulSet 详细信息：

```
kubectl -n ${namespace} describe sts ${name}
```

11.4.1.2 更新资源

- 为 TiDBCluster 增加 Annotation：

```
kubectl -n ${namespace} annotate tc ${cluster_name} ${key}=${value}
```

为 TiDBCluster 增加强制升级 Annotation：

```
kubectl -n ${namespace} annotate --overwrite tc ${cluster_name} tidb.
    ↪ pingcap.com/force-upgrade=true
```

为 TiDBCluster 删除强制升级 Annotation：

```
kubectl -n ${namespace} annotate tc ${cluster_name} tidb.pingcap.com/
    ↪ force-upgrade-
```

为 Pod 开启 Debug 模式：

```
kubectl -n ${namespace} annotate pod ${pod_name} runmode=debug
```

11.4.1.3 编辑资源

- 编辑 TidbCluster：

```
kubectl -n ${namespace} edit tc ${name}
```

11.4.1.4 Patch 资源

- Patch PV ReclaimPolicy：

```
kubectl patch pv ${name} -p '{"spec":{"persistentVolumeReclaimPolicy":"
    ↪ Delete"}}'
```

- Patch PVC：

```
kubectl -n ${namespace} patch pvc ${name} -p '{"spec": {"resources": {"
    ↪ requests": {"storage": "100Gi"}}}'
```

- Patch StorageClass：

```
kubectl patch storageclass ${name} -p '{"allowVolumeExpansion": true}'
```

11.4.1.5 创建资源

- 通过 Yaml 文件创建集群：

```
kubectl -n ${namespace} apply -f ${file}
```

- 创建 Namespace：

```
kubectl create ns ${namespace}
```

- 创建 Secret：

  创建证书的 Secret：

```
kubectl -n ${namespace} create secret generic ${secret_name} --from-
    ↪ file=tls.crt=${cert_path} --from-file=tls.key=${key_path} --from-
    ↪ file=ca.crt=${ca_path}
```

  创建用户名、密码的 Secret：

```
kubectl -n ${namespace} create secret generic ${secret_name} --from-
    ↪ literal=user=${user} --from-literal=password=${password}
```

11.4.1.6 与 Running Pod 交互

- 查看 PD 配置文件：

```
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/pd/pd.toml
```

- 查看 TiDB 配置文件：

```
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/tidb/tidb.toml
```

- 查看 TiKV 配置文件：

```
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/tikv/tikv.toml
```

- 查看 Pod Log：

```
kubectl -n ${namespace} logs ${pod_name} -f
```

  查看上一次容器的 Log：

```
kubectl -n ${namespace} logs ${pod_name} -p
```

  如果 Pod 内有多个容器，查看某一个容器的 Log：

```
kubectl -n ${namespace} logs ${pod_name} -c ${container_name}
```

- 暴露服务：

```
kubectl -n ${namespace} port-forward svc/${service_name} ${local_port}:
    ↪ ${port_in_pod}
```

暴露 PD 服务：

```
kubectl -n ${namespace} port-forward svc/${cluster_name}-pd 2379:2379
```

11.4.1.7 与 Node 交互

- 把 Node 设置为不可调度：

```
kubectl cordon ${node_name}
```

- 取消 Node 不可调度：

```
kubectl uncordon ${node_name}
```

11.4.1.8 删除资源

- 删除 Pod：

```
kubectl delete -n ${namespace} pod ${pod_name}
```

- 删除 PVC：

```
kubectl delete -n ${namespace} pvc ${pvc_name}
```

- 删除 TidbCluster：

```
kubectl delete -n ${namespace} tc ${tc_name}
```

- 删除 TidbMonitor：

```
kubectl delete -n ${namespace} tidbmonitor ${tidb_monitor_name}
```

- 删除 TidbClusterAutoScaler：

```
kubectl -n ${namespace} delete tidbclusterautoscaler ${name}
```

11.4.1.9 更多

其他更多 kubectl 的使用，请参考 Kubectl Cheat Sheet。

11.4.2 Helm

11.4.2.1 添加 Helm Repo

```
helm repo add pingcap https://charts.pingcap.org/
```

11.4.2.2 更新 Helm Repo

```
helm repo update
```

11.4.2.3 查看可用的 Helm Chart

- 查看 Helm Hub 中的 Chart

  ```
  helm search hub ${chart_name}
  ```

  示例：

  ```
  helm search hub mysql
  ```

- 查看其他 Repo 中的 Chart

  ```
  helm search repo ${chart_name} -l --devel
  ```

  示例：

  ```
  helm search repo tidb-operator -l --devel
  ```

11.4.2.4 获取 Helm Chart 默认 values.yaml

```
helm inspect values ${chart_name} --version=${chart_version} > values.yaml
```

示例：

```
helm inspect values pingcap/tidb-operator --version=v1.1.15 > values-tidb-
    ↪ operator.yaml
```

11.4.2.5 使用 Helm Chart 部署

```
helm install ${name} ${chart_name} --namespace=${namespace} --version=${
    ↪ chart_version} -f ${values_file}
```

示例：

```
helm install tidb-operator pingcap/tidb-operator --namespace=tidb-admin --
    ↪ version=v1.1.15 -f values-tidb-operator.yaml
```

11.4.2.6 查看已经部署的 Helm Release

```
helm ls
```

11.4.2.7 升级 Helm Release

```
helm upgrade ${name} ${chart_name} --version=${chart_version} -f ${
    ↪ values_file}
```

示例：

```
helm upgrade tidb-operator pingcap/tidb-operator --version=v1.1.15 -f values
    ↪ -tidb-operator.yaml
```

11.4.2.8 删除 Helm Release

```
helm uninstall ${name} -n ${namespace}
```

示例：

```
helm uninstall tidb-operator -n tidb-admin
```

11.4.2.9 更多

其他更多 Helm 的使用，请参考 Helm Commands。

11.5 TiDB Operator 需要的 RBAC 规则

Kubernetes 基于角色的访问控制 (RBAC) 规则是通过 Role 或者 ClusterRole 来进行管理的，并通过 RoleBinding 或 ClusterRoleBinding 将其权限赋予一个或者一组用户。

11.5.1 Cluster 级别管理 TiDB 集群

部署 TiDB Operator 时默认设置了 `clusterScoped=true`，TiDB Operator 能管理 Kubernetes 集群内所有 TiDB 集群。

要查看为 TiDB Operator 创建的 ClusterRole，请使用以下命令：

```
kubectl get clusterrole | grep tidb
```

输出结果如下：

```
tidb-operator:tidb-controller-manager                              2021-05-04T13
    ↪ :08:55Z
tidb-operator:tidb-scheduler                                       2021-05-04T13
    ↪ :08:55Z
```

其中：

- `tidb-operator:tidb-controller-manager` 是为 `tidb-controller-manager` Pod 创建的 ClusterRole。
- `tidb-operator:tidb-scheduler` 是为 `tidb-scheduler` Pod 创建的 ClusterRole。

11.5.1.1 tidb-controller-manager ClusterRole 权限

以下表格列出了 tidb-controller-manager ClusterRole 对应的权限。

资源	非资源 URLs	资源名	动作	解释
events	-	-	[*]	输出 Event 信息
services	-	-	[*]	操作 Service 资源
statefulsets.apps.pingcap.com/status			[*]	AdvancedStatefulSet=true 时，需要操作此资源，详细信息可以参考增强型 StatefulSet 控制器
statefulsets.apps.pingcap.com	-	-	[*]	AdvancedStatefulSet=true 时，需要操作此资源，详细信息可以参考增强型 StatefulSet 控制器

资源	非资源 URLs	资源名	动作	解释
controllerrevisions.apps		-	[*]	Kubernetes StatefulSet/DaemonSet 版本控制
deployments.apps		-	[*]	操作 Deployment 资源
statefulsets.apps		-	[*]	操作 Statefulset 资源
ingresses.extensions		-	[*]	操作监控系统 Ingress 资源
*.pingcap.com		-	[*]	操作 pingcap.com 下所有自定义资源
configmaps	-	-	[create get list watch update delete]	操作 ConfigMap 资源
endpoints	-	-	[create get list watch update delete]	操作 Endpoints 资源

资源	非资源 URLs	资源名	动作	解释
serviceaccounts	-	-	[create get update delete]	为 Tidb-Monitor/Discovery 服务创建 ServiceAccount
clusterrolebindings.rbac.authorization.k8s.io	-	-	[create get update delete]	为 TidbMonitor 服务创建 ClusterRoleBinding
rolebindings.rbac.authorization.k8s.io	-	-	[create get update delete]	为 Tidb-Monitor/Discovery 服务创建 RoleBinding
secrets	-	-	[create update get list watch delete]	操作 Secret 资源
clusterroles.rbac.authorization.k8s.io	-	-	[escalate create get update delete]	为 TidbMonitor 服务创建 ClusterRole

资源	非资源 URLs	资源名	动作	解释
roles.rbac.authorization.k8s.io	-	-	[escalate create get update delete]	为 Tidb-Monitor/Discovery 服务创建 Role
persistentvolumeclaims		-	[get list watch create update delete patch]	操作 PVC 资源
jobs.batch	-	-	[get list watch create update delete]	TiDB 集群初始化、备份、恢复操作使用 Job 进行
persistentvolumes		-	[get list watch patch update]	为 PV 添加集群信息相关 Label、修改 `persistentVolumeReclaimPolicy` ↪ 等操作
pods	-	-	[get list watch update delete]	操作 Pod 资源

资源	非资源 URLs	资源名	动作	解释
nodes	-	-	[get list watch]	读取 Node Label 并根据 Label 信息为 TiKV、TiFlash 设置 Store Label
storageclasses.storage.k8s.io		-	[get list watch]	扩展 PVC 存储之前确认 Storage-Class 是否支持 VolumeExpansion
-	[/metrics]	-	[get]	读取监控指标

注意：

- 在非资源 **URLs** 列中，- 表示该项没有非资源 URLs。
- 在资源名列中，- 表示该项没有资源名。
- 在动作列中，* 表示支持 Kubernetes 集群支持的所有动作。

11.5.1.2 tidb-scheduler ClusterRole 权限

以下表格列出了 tidb-scheduler ClusterRole 对应的权限。

资源	非资源 URLs	资源名	动作	解释
leases.coordination.k8s.io		-	[create]	Leader 选举需要创建 Lease 资源锁

资源	非资源 URLs	资源名	动作	解释
endpoints	-	-	[delete get patch update]	操作 Endpoints 资源
persistentvolumeclaims		-	[get list update]	读取 PD/TiKV PVC 信息，更新调度信息到 PVC Label
configmaps	-	-	[get list watch]	读取 ConfigMap 资源
pods	-	-	[get list watch]	读取 Pod 信息
nodes	-	-	[get list]	读取 Node 信息
leases.coordination.k8s.io		[tidb-scheduler]	[get update]	Leader 选举需要读取/更新 Lease 资源锁
tidbclusters.pingcap.com		-	[get]	读取 Tidb-cluster 信息

注意：

- 在非资源 URLs 列中，- 表示该项没有非资源 URLs。在资源名列中，- 表示该项没有资源名。

11.5.2 Namespace 级别管理 TiDB 集群

如果部署 TiDB Operator 时设置了 `clusterScoped=false`，TiDB Operator 将在 Namespace 级别管理 TiDB 集群。

- 要查看为 TiDB Operator 创建的 ClusterRole，请使用以下命令：

```
kubectl get clusterrole | grep tidb
```

输出结果如下：

```
tidb-operator:tidb-controller-manager
    ↪ 2021-05-04T13:08:55Z
```

`tidb-operator:tidb-controller-manager` 是为 `tidb-controller-manager` Pod 创建的 ClusterRole。

> **注意：**
> 如果部署 TiDB Operator 时已设置 `controllerManager.clusterPermissions.nodes`、`controllerManager.clusterPermissions.persistentvolumes`、`controllerManager.clusterPermissions.storageclasses` 都为 `false`，则不会创建该 ClusterRole。

- 要查看为 TiDB Operator 创建的 Role，请使用以下命令：

```
kubectl get role -n tidb-admin
```

输出结果如下：

```
tidb-admin   tidb-operator:tidb-controller-manager      2021-05-04T13
    ↪ :08:55Z
tidb-admin   tidb-operator:tidb-scheduler              2021-05-04T13
    ↪ :08:55Z
```

其中：

  - `tidb-operator:tidb-controller-manager` 是为 `tidb-controller-manager` Pod 创建的 Role。
  - `tidb-operator:tidb-scheduler` 是为 `tidb-scheduler` Pod 创建的 Role。

11.5.2.1 tidb-controller-manager ClusterRole 权限

以下表格列出了 tidb-controller-manager ClusterRole 对应的权限。

资源	非资源 URLs	资源名	动作	解释
persistentvolumes		-	[get list watch patch update]	为 PV 添加集群信息相关 Label、修改 persistentVolumeReclaimPolicy ↪ 等操作
nodes	-	-	[get list watch]	读取 Node Label 并根据 Label 信息为 TiKV、TiFlash 设置 Store Label
storageclasses.storage.k8s.io		-	[get list watch]	扩展 PVC 存储之前确认 Storage-Class 是否支持 VolumeExpansion ↪

注意：

- 在非资源 **URLs** 列中，- 表示该项没有非资源 URLs。
- 在资源名列中，- 表示该项没有资源名。

11.5.2.2 tidb-controller-manager Role 权限

以下表格列出了 tidb-controller-manager Role 对应的权限。

资源	非资源 URLs	资源名	动作	解释
events	-	-	[*]	输出 Event 信息
services	-	-	[*]	操作 Service 资源
statefulsets.apps.pingcap.com/status			[*]	AdvancedStatefulSet=true 时，需要操作此资源，详细信息可以参考增强型 StatefulSet 控制器
statefulsets.apps.pingcap.com	-	-	[*]	AdvancedStatefulSet=true 时，需要操作此资源，详细信息可以参考增强型 StatefulSet 控制器
controllerrevisions.apps	-	-	[*]	Kubernetes StatefulSet/DaemonSet 版本控制
deployments.apps	-	-	[*]	操作 Deployment 资源
statefulsets.apps	-	-	[*]	操作 Statefulset 资源

资源	非资源 URLs	资源名	动作	解释
ingresses.extensions		-	[*]	操作监控系统 Ingress 资源
*.pingcap.com	-	-	[*]	操作 pingcap.com 下所有自定义资源
configmaps	-	-	[create get list watch update delete]	操作 ConfigMap 资源
endpoints	-	-	[create get list watch update delete]	操作 Endpoints 资源
serviceaccounts	-	-	[create get update delete]	为 Tidb-Monitor/Discovery 服务创建 ServiceAccount
rolebindings.rbac.authorization.k8s.io			[create get update delete]	为 Tidb-Monitor/Discovery 服务创建 RoleBinding

资源	非资源 URLs	资源名	动作	解释
secrets	-	-	[create update get list watch delete]	操作 Secret 资源
roles.rbac.authorization.k8s.io	-	-	[escalate create get update delete]	为 Tidb-Monitor/Discovery 服务创建 Role
persistentvolumeclaims		-	[get list watch create update delete patch]	操作 PVC 资源
jobs.batch	-	-	[get list watch create update delete]	TiDB 集群初始化、备份、恢复操作使用 Job 进行
pods	-	-	[get list watch update delete]	操作 Pod 资源

注意：

- **在非资源 URLs 列中，- 表示该项没有非资源 URLs。**

- 在资源名列中，- 表示该项没有资源名。
- 在动作列中，* 表示支持 Kubernetes 集群支持的所有动作。

11.5.2.3 tidb-scheduler Role 权限

以下表格列出了 tidb-scheduler Role 对应的权限。

资源	非资源 URLs	资源名	动作	解释
leases.coordination.k8s.io		-	[create]	leader 选举需要创建 Lease 资源锁
endpoints	-	-	[delete get patch update]	操作 Endpoints 资源
persistentvolumeclaims		-	[get list update]	读取 PD/TiKV PVC 信息，更新调度信息到 PVC Label
configmaps	-	-	[get list watch]	读取 Configmap 资源
pods	-	-	[get list watch]	读取 Pod 信息
nodes	-	-	[get list]	读取 Node 信息
leases.coordination.k8s.io		[tidb-scheduler]	[get update]	leader 选举需要读取/更新 Lease 资源锁

资源	非资源 URLs	资源名	动作	解释
tidbclusters.pingcap.com		-	[get]	读取 Tidb-cluster 信息

注意：

- 在非资源 **URLs** 列中，- 表示该项没有非资源 URLs。
- 在资源名列中，- 表示该项没有资源名。

11.6 工具

11.6.1 tkctl 使用指南

`tkctl` (TiDB Kubernetes Control) 是为 TiDB in Kubernetes 设计的命令行工具，用于运维集群和诊断集群问题。

注意：

PingCAP 从 v1.1.x 开始不再维护 `tkctl`，以下部分功能可能不可用，请直接使用对应的 `kubectl` 命令。

11.6.1.1 安装

安装 `tkctl` 时，可以直接下载预编译的可执行文件，也可以自行从源码进行编译。

11.6.1.1.1 下载预编译的可执行文件

- MacOS
- Linux
- Windows

下载解压后，将 `tkctl` 可执行文件加入到可执行文件路径 (`PATH`) 中即完成安装。

11.6.1.1.2 源码编译

要求：Go 版本 1.11 及以上

```
git clone https://github.com/pingcap/tidb-operator.git && \
GOOS=${YOUR_GOOS} make cli && \
mv tkctl /usr/local/bin/tkctl
```

11.6.1.2 命令自动补全

你可以配置 tkctl 的自动补全以简化使用。

为 BASH 配置自动补全（需要预先安装 bash-completion）的方法如下。

在当前 shell 中设置自动补全：

```
source <(tkctl completion bash)
```

永久设置自动补全：

```
echo "if hash tkctl 2>/dev/null; then source <(tkctl completion bash); fi"
    ↪ >> ~/.bashrc
```

为 ZSH 配置自动补全的方法如下。

在当前 shell 中设置自动补全：

```
source <(tkctl completion zsh)
```

永久设置自动补全：

```
echo "if hash tkctl 2>/dev/null; then source <(tkctl completion zsh); fi" >>
    ↪  ~/.zshrc
```

11.6.1.3 Kubernetes 配置

tkctl 复用了 kubeconfig 文件（默认位置是 ~/.kube/config）来连接 Kubernetes 集群。你可以通过下面的命令来验证 kubeconfig 是否设置正确：

```
tkctl version
```

假如上面的命令正确输出服务端的 TiDB Operator 版本，则 kubeconfig 配置正确。

11.6.1.4 所有命令

11.6.1.4.1 tkctl version

该命令用于展示本地 tkctl 和集群中 tidb-operator 的版本：

示例如下：

```
tkctl version
```

```
Client Version: v1.0.0-beta.1-p2-93-g6598b4d3e75705-dirty
TiDB Controller Manager Version: pingcap/tidb-operator:latest
TiDB Scheduler Version: pingcap/tidb-operator:latest
```

11.6.1.4.2 tkctl list

该命令用于列出所有已安装的 TiDB 集群：

参数	缩写	说明
–all-namespaces	-A	是否查询所有的 Kubernetes Namespace
–output	-o	输出格式，可选值有 [default,json,yaml]，默认值为 default

示例如下：

```
tkctl list -A
```

```
NAMESPACE NAME         PD    TIKV   TIDB   AGE
foo       demo-cluster 3/3   3/3    2/2    11m
bar       demo-cluster 3/3   3/3    1/2    11m
```

11.6.1.4.3 tkctl use

该命令用于指定当前 tkctl 操作的 TiDB 集群，在使用该命令设置当前操作的 TiDB 集群后，所有针对集群的操作命令会自动选定该集群，从而可以略去 --tidbcluster 参数。

示例如下：

```
tkctl use --namespace=foo demo-cluster
```

```
Tidb cluster switched to foo/demo-cluster
```

11.6.1.4.4 tkctl info

该命令用于展示 TiDB 集群的信息。

参数	缩写	说明
–tidb-cluster	-t	指定 TiDB 集群，默认为当前使用的 TiDB 集群

示例如下：

```
tkctl info
```

```
Name:             demo-cluster
Namespace:        foo
CreationTimestamp: 2019-04-17 17:33:41 +0800 CST
Overview:
        Phase   Ready Desired CPU   Memory Storage Version
        -----   ----- ------- ---   ------ ------- -------
  PD:   Normal  3     3       200m  1Gi    1Gi     pingcap/pd:v3.0.0-rc.1
  TiKV: Normal  3     3       1000m 2Gi    10Gi    pingcap/tikv:v3.0.0-rc.1
  TiDB  Upgrade 1     2       500m  1Gi            pingcap/tidb:v3.0.0-rc.1
Endpoints(NodePort):
  - 172.16.4.158:31441
  - 172.16.4.155:31441
```

11.6.1.4.5 tkctl get [component]

该命令用于获取 TiDB 集群中组件的详细信息。

可选的组件 (component) 有：pd、tikv、tidb、volume 和 all（用于同时查询所有组件）。

参数	缩写	说明
–tidb-cluster	-t	指定 TiDB 集群，默认为当前使用的 TiDB 集群
–output	-o	输出格式，可选值有 default、json 和 yaml，默认值为 default

示例如下：

```
tkctl get tikv
```

```
NAME                READY   STATUS    MEMORY          CPU   RESTARTS  AGE     NODE
demo-cluster-tikv-0 2/2     Running   2098Mi/4196Mi   2/2   0         3m19s
    ↪ 172.16.4.155
demo-cluster-tikv-1 2/2     Running   2098Mi/4196Mi   2/2   0         4m8s
    ↪ 172.16.4.160
demo-cluster-tikv-2 2/2     Running   2098Mi/4196Mi   2/2   0         4m45s
    ↪ 172.16.4.157
```

```
tkctl get volume
```

```
VOLUME            CLAIM                     STATUS  CAPACITY  NODE
    ↪ LOCAL
```

```
local-pv-d5dad2cf tikv-demo-cluster-tikv-0 Bound 1476Gi    172.16.4.155 /mnt
    ↪ /disks/local-pv56
local-pv-5ade8580 tikv-demo-cluster-tikv-1 Bound 1476Gi    172.16.4.160 /mnt
    ↪ /disks/local-pv33
local-pv-ed2ffe50 tikv-demo-cluster-tikv-2 Bound 1476Gi    172.16.4.157 /mnt
    ↪ /disks/local-pv13
local-pv-74ee0364 pd-demo-cluster-pd-0   Bound   1476Gi    172.16.4.155 /mnt
    ↪ /disks/local-pv46
local-pv-842034e6 pd-demo-cluster-pd-1   Bound   1476Gi    172.16.4.158 /mnt
    ↪ /disks/local-pv74
local-pv-e54c122a pd-demo-cluster-pd-2   Bound   1476Gi    172.16.4.156 /mnt
    ↪ /disks/local-pv72
```

11.6.1.4.6 tkctl debug [pod_name]

该命令用于诊断 TiDB 集群中的 Pod。

实际使用时，该命令会在目标 Pod 的宿主机上启动 debug launcher Pod 并以指定镜像启动一个 debug 容器，该容器会与目标 Pod 中的容器共享 namespace，因此可以无缝使用 debug 容器中的各种工具对目标容器进行诊断。

参数	缩写	描述
–image		指定 debug 容器使用的镜像，默认为 pingcap/tidb-debug:latest
–launcher-image		指定启动 debug 容器的 debug launcher Pod 使用的镜像，默认为 pingcap/debug-launcher:latest
–container	-c	选择需要诊断的容器，默认为 Pod 定义中的第一个容器

参数	缩写	描述
–docker-socket		指定目标节点上的 Docker Socket，默认为 /var/run/↪ docker.↪ sock
–privileged		是否为 debug 容器开启 privileged 模式

注意：

Debug 容器使用的默认镜像包含了绝大多数的诊断工具，因此体积较大，假如只需要 pd-ctl 和 tidb-ctl，可以使用 --image=pingcap/tidb-control↪ :latest 来指定使用 tidb-control 镜像。

示例如下：

```
tkctl debug demo-cluster-tikv-0
```

```
ps -ef
```

由于 debug 容器和目标容器拥有不同的根文件系统，在 tidb-debug 容器中使用 GDB 和 perf 等工具时可能会碰到一些问题，下面将补充说明如何解决这些问题。

GDB

使用 GDB 调试目标容器中的进程时，需要将 program 参数设置为目标容器中的可执行文件。假如是在 tidb-debug 以外的其它 debug 容器中进行调试，或者调试的目标进行 pid 不为 1，则需要使用 set sysroot 命令调整动态链接库的加载位置。操作如下：

```
tkctl debug demo-cluster-tikv-0
```

```
gdb /proc/${pid:-1}/root/tikv-server 1
```

tidb-debug 中预配置的 .gdbinit 会将 sysroot 设置为 /proc/1/root/，因此在 tidb↪ -debug 中，假如目标容器的 pid 为 1，则下面的命令可以省略。

```
(gdb) set sysroot /proc/${pid}/root/
```

开始调试：

```
(gdb) thread apply all bt
```

```
(gdb) info threads
```

Perf 以及火焰图

使用 perf 命令和 run-flamegraph.sh 脚本时，需要将目标容器的可执行文件拷贝到 Debug 容器中：

```
tkctl debug demo-cluster-tikv-0
```

```
cp /proc/1/root/tikv-server /
```

```
./run_flamegraph.sh 1
```

该脚本会自动将生成的火焰图（SVG 格式）上传至 transfer.sh，通过访问脚本输出的链接即可下载火焰图。

11.6.1.4.7 tkctl ctop

命令的完整形式：`tkctl ctop [pod_name | node/node_name ]`。

该命令用于查看集群中 Pod 或 Node 的实时监控信息，和 `kubectl top` 相比，`tkctl ctop` 还会展示网络和磁盘的使用信息。

参数	简写	描述
–image		指定 ctop 的镜像，默认为 quay.io/vektorlab/ctop:0.7.2
–docker-socket		指定 ctop 使用的 Docker Socket，默认为 /var/run/docker.sock

示例如下：

```
tkctl ctop demo-cluster-tikv-0
```

```
tkctl ctop node/172.16.4.155
```

11.6.1.4.8 tkctl help [command]

该命令用于展示各个子命令的帮助信息。

示例如下：

```
tkctl help debug
```

11.6.1.4.9 tkctl options

该命令用于展示 `tkctl` 的所有的全局参数。

示例如下：

```
tkctl options
```

```
The following options can be passed to any command:

      --alsologtostderr=false: log to standard error as well as files
      --as='': Username to impersonate for the operation
      --as-group=[]: Group to impersonate for the operation, this flag can
         ↪ be repeated to specify multiple groups.
      --cache-dir='/Users/alei/.kube/http-cache': Default HTTP cache
         ↪ directory
      --certificate-authority='': Path to a cert file for the certificate
         ↪ authority
      --client-certificate='': Path to a client certificate file for TLS
      --client-key='': Path to a client key file for TLS
      --cluster='': The name of the kubeconfig cluster to use
      --context='': The name of the kubeconfig context to use
      --insecure-skip-tls-verify=false: If true, the server's certificate
         ↪ will not be checked for validity. This will
make your HTTPS connections insecure
      --kubeconfig='': Path to the kubeconfig file to use for CLI requests.
      --log_backtrace_at=:0: when logging hits line file:N, emit a stack
         ↪ trace
      --log_dir='': If non-empty, write log files in this directory
      --logtostderr=true: log to standard error instead of files
  -n, --namespace='': If present, the namespace scope for this CLI request
      --request-timeout='0': The length of time to wait before giving up on
         ↪ a single server request. Non-zero values
should contain a corresponding time unit (e.g. 1s, 2m, 3h). A value of zero
   ↪ means don't timeout requests.
  -s, --server='': The address and port of the Kubernetes API server
      --stderrthreshold=2: logs at or above this threshold go to stderr
  -t, --tidbcluster='': Tidb cluster name
      --token='': Bearer token for authentication to the API server
      --user='': The name of the kubeconfig user to use
  -v, --v=0: log level for V logs
      --vmodule=: comma-separated list of pattern=N settings for file-
         ↪ filtered logging
```

这些参数主要用于指定如何连接 Kubernetes 集群，其中最常用的参数是：

- `--context`：指定目标 Kubernetes 集群

- `--namespace`：指定 Namespace

11.6.2 Kubernetes 上的 TiDB 工具指南

Kubernetes 上的 TiDB 运维管理需要使用一些开源工具。同时，在 Kubernetes 上使用 TiDB 生态工具时，也有特殊的操作要求。本文档详细描述 Kubernetes 上的 TiDB 相关的工具及其使用方法。

11.6.2.1 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` 打开本地到 PD 服务的连接：

```
kubectl port-forward -n ${namespace} svc/${cluster_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` 访问到 PD 服务，从而直接使用 `pd-ctl` 命令的默认参数执行操作，如：

```
pd-ctl -d config show
```

假如你本地的 2379 被占据，则需要选择其它端口：

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-pd ${local_port
    ↪ }:2379 &>/tmp/portforward-pd.log &
```

此时，需要为 `pd-ctl` 命令显式指定 PD 端口：

```
pd-ctl -u 127.0.0.1:${local_port} -d config show
```

11.6.2.2 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，针对 TiKV Control 的不同操作模式，有不同的操作步骤。

- 远程模式：此模式下 `tikv-ctl` 命令需要通过网络访问 TiKV 服务或 PD 服务，因此需要先使用 `kubectl port-forward` 打开本地到 PD 服务以及目标 TiKV 节点的连接：

  ```
  kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379
      ↪ &>/tmp/portforward-pd.log &
  ```

  ```
  kubectl port-forward -n ${namespace} ${pod_name} 20160:20160 &>/tmp/
      ↪ portforward-tikv.log &
  ```

  打开连接后，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 ${subcommands}
```

```
tikv-ctl --pd 127.0.0.1:2379 compact-cluster
```

- 本地模式：本地模式需要访问 TiKV 的数据文件，并且需要停止正在运行的 TiKV 实例。需要先使用诊断模式关闭 TiKV 实例自动重启，关闭 TiKV 进程，再进入目标 TiKV Pod 中使用 tikv-ctl 来执行操作，步骤如下：

  1. 进入诊断模式：

     ```
     kubectl annotate pod ${pod_name} -n ${namespace} runmode=debug
     ```

  2. 关闭 TiKV 进程：

     ```
     kubectl exec ${pod_name} -n ${namespace} -c tikv -- kill -s TERM 1
     ```

  3. 等待 TiKV 容器重启后进入容器：

     ```
     kubectl exec -it ${pod_name} -n ${namespace} -- sh
     ```

  4. 开始使用 tikv-ctl 的本地模式，TiKV 容器中的默认 db 路径是 /var/lib/tikv/db：

     ```
     ./tikv-ctl --data-dir /var/lib/tikv size -r 2
     ```

11.6.2.3 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 打开到集群中 TiDB 节点和 PD 服务的连接：

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379 &>/tmp
    ↪ /portforward-pd.log &
```

```
kubectl port-forward -n ${namespace} ${pod_name} 10080:10080 &>/tmp/
    ↪ portforward-tidb.log &
```

接下来便可开始使用 tidb-ctl 命令：

```
tidb-ctl schema in mysql
```

11.6.2.4 使用 Helm

Helm 是一个 Kubernetes 的包管理工具。安装步骤如下：

11.6.2.4.1 安装 Helm

参考官方文档安装 Helm。

如果服务器没有外网，需要先将 Helm 在有外网的机器上下载下来，然后再拷贝到服务器上，这里以安装 Helm 3.4.1 为例：

```
wget https://get.helm.sh/helm-v3.4.1-linux-amd64.tar.gz
tar zxvf helm-v3.4.1-linux-amd64.tar.gz
```

解压之后，有以下文件：

```
linux-amd64/
linux-amd64/README.md
linux-amd64/helm
linux-amd64/LICENSE
```

请自行将 linux-amd64/helm 文件拷贝到服务器上，并将其放到 /usr/local/bin/ 目录下即可。

然后执行 helm verison，如果正常输出则表示 Helm 安装成功：

```
helm version
```

```
version.BuildInfo{Version:"v3.4.1", GitCommit:"
    ↪ c4e74854886b2efe3321e185578e6db9be0a6e29", GitTreeState:"clean",
    ↪ GoVersion:"go1.14.11"}
```

11.6.2.4.2 配置 Helm repo

Kubernetes 应用在 Helm 中被打包为 chart。PingCAP 针对 Kubernetes 上的 TiDB 部署运维提供了多个 Helm chart：

- tidb-operator：用于部署 TiDB Operator；
- tidb-cluster：用于部署 TiDB 集群；
- tidb-backup：用于 TiDB 集群备份恢复；
- tidb-lightning：用于 TiDB 集群导入数据；
- tidb-drainer：用于部署 TiDB Drainer；
- tikv-importer：用于部署 TiKV Importer；

这些 chart 都托管在 PingCAP 维护的 helm chart 仓库 https://charts.pingcap.org/ 中，你可以通过下面的命令添加该仓库：

```
helm repo add pingcap https://charts.pingcap.org/
```

添加完成后，可以使用 helm search 搜索 PingCAP 提供的 chart：

```
helm search repo pingcap
```

```
NAME                    CHART VERSION APP VERSION   DESCRIPTION
pingcap/tidb-backup  v1.1.15                         A Helm chart for TiDB
    ↪ Backup or Restore
pingcap/tidb-cluster v1.1.15                         A Helm chart for TiDB
    ↪ Cluster
pingcap/tidb-drainer v1.1.15                         A Helm chart for TiDB
    ↪ Binlog drainer.
pingcap/tidb-lightning v1.1.15                       A Helm chart for TiDB
    ↪ Lightning
pingcap/tidb-operator v1.1.15       v1.1.15          tidb-operator Helm chart
    ↪ for Kubernetes
pingcap/tikv-importer v1.1.15                        A Helm chart for TiKV
    ↪ Importer
```

当新版本的 chart 发布后，你可以使用 `helm repo update` 命令更新本地对于仓库的缓存：

```
helm repo update
```

11.6.2.4.3 Helm 常用操作

Helm 的常用操作有部署（`helm install`）、升级（`helm upgrade`）、销毁（`helm uninstall`）、查询（`helm ls`）。Helm chart 往往都有很多可配置参数，通过命令行进行配置比较繁琐，因此推荐使用 YAML 文件的形式来编写这些配置项。基于 Helm 社区约定俗称的命名方式，在文档中将用于配置 chart 的 YAML 文件称为 `values.yaml` 文件。

执行部署、升级、销毁等操作前，可以使用 `helm ls` 查看集群中已部署的应用：

```
helm ls
```

在执行部署和升级操作时，必须指定使用的 chart 名字（`chart-name`）和部署后的应用名（`release-name`），还可以指定一个或多个 `values.yaml` 文件来配置 chart。此外，假如对 chart 有特定的版本需求，则需要通过 `--version` 参数指定 `chart-version`（默认为最新的 GA 版本）。命令形式如下：

- 执行安装：

```
helm install ${release_name} ${chart_name} --namespace=${namespace} --
    ↪ version=${chart_version} -f ${values_file}
```

- 执行升级（升级可以是修改 `chart-version` 升级到新版本的 chart，也可以是修改 `values.yaml` 文件更新应用配置）：

```
helm upgrade ${release_name} ${chart_name} --version=${chart_version} -
    ↪ f ${values_file}
```

最后，假如要删除 helm 部署的应用，可以执行：

```
helm uninstall ${release_name} -n ${namespace}
```

更多 helm 的相关文档，请参考 Helm 官方文档。

11.6.2.4.4 离线情况下使用 Helm chart

如果服务器上没有外网，就无法通过配置 Helm repo 来安装 TiDB Operator 组件以及其他应用。这时，需要在有外网的机器上下载集群安装需用到的 chart 文件，再拷贝到服务器上。

通过以下命令，下载集群安装时需要的 chart 文件：

```
wget http://charts.pingcap.org/tidb-operator-v1.1.15.tgz
wget http://charts.pingcap.org/tidb-drainer-v1.1.15.tgz
wget http://charts.pingcap.org/tidb-lightning-v1.1.15.tgz
```

将这些 chart 文件拷贝到服务器上并解压，可以通过 `helm install` 命令使用这些 chart 来安装相应组件，以 `tidb-operator` 为例：

```
tar zxvf tidb-operator.v1.1.15.tgz
helm install ${release_name} ./tidb-operator --namespace=${namespace}
```

11.6.2.5 使用 Terraform

Terraform 是一个基础设施即代码 (Infrastructure as Code) 管理工具。它允许用户使用声明式的风格描述自己的基础设施，并针对描述生成执行计划来创建或调整真实世界的计算资源。Kubernetes 上的 TiDB 使用 Terraform 来在公有云上创建和管理 TiDB 集群。

你可以参考 Terraform 官方文档 来安装 Terraform。

11.7 配置

11.7.1 Kubernetes 上的 TiDB Binlog Drainer 配置

本文档介绍 Kubernetes 上 TiDB Binlog drainer 的配置参数。

11.7.1.1 配置参数

下表包含所有用于 `tidb-drainer` chart 的配置参数。关于如何配置这些参数，可参阅使用 Helm。

参数	说明	默认值
`timezone`	时区配置	`UTC`

参数	说明	默认值
drainerName	Statefulset 名称	""
clusterName	TiDB 集群的名称源	demo
clusterVersion	TiDB 集群的版本源	v3.0.1
baseImage	TiDB Binlog 的基础镜像	pingcap/tidb-binlog
imagePullPolicy	镜像的拉取策略	IfNotPresent
logLevel	drainer 进程的日志级别	info

参数	说明	默认值
storageClassName ↪	所使用的 ↪ storageClass ↪ 。storageClassName ↪ 是 Kubernetes 集群提供的一种存储，可以映射到服务质量级别、备份策略或集群管理员确定的任何策略。详情可参阅 storage-classes	local-storage

参数	说明	默认值
storage	drainer Pod 的存储限制。请注意，如果 db-type 设为 pd，则应将本参数值设得大一些	10Gi
disableDetect	决定是否禁用事故检测	false
initialCommitTs	如果 drainer 没有断点，则用于初始化断点。该参数值为 string 类型，如 "424364429251444742"	"-1"

参数	说明	默认值
tlsCluster. enableTLS	是否开启集群间 TLS	false
config	传递到 drainer 的配置文件。详情可参阅 drainer.toml	（见下文）
resources	drainer Pod 的资源限制和请求	{}
nodeSelector	确保 drainer Pod 仅被调度到具有特定键值对作为标签的节点上。详情可参阅 nodes-elector	{}

参数	说明	默认值
tolerations	适用于 drainer Pod，允许将 Pod 调度到有指定 taint 的节点上。详情可参阅 taint-and-toleration	{}
affinity	定义 drainer Pod 的调度策略和首选项。详情可参阅 affinity-and-anti-affinity	{}

config 的默认值为：

```
detect-interval = 10
compressor = ""
[syncer]
worker-count = 16
disable-dispatch = false
ignore-schemas = "INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql"
```

```
safe-mode = false
txn-batch = 20
db-type = "file"
[syncer.to]
dir = "/data/pb"
```

11.7.2 tidb-cluster chart 配置

本文介绍 tidb-cluster chart 配置。

> **注意：**
>
> 对于 TiDBOperator v1.1 及以上版本，不再建议使用 tidb-cluster chart 部署、管理 TiDB 集群，详细信息请参考TiDB Operator v1.1 重要注意事项。

11.7.2.1 配置参数

参数名	说明	默认值
`rbac.create`	是否启用 Kubernetes 的 RBAC	`true`

参数名	说明	默认值
clusterName	TiDB 集群名，默认不设置该变量，tidb-cluster 会直接用执行安装时的 ReleaseName 代替	nil
extraLabels	添加额外的 labels 到 TidbCluster 对象 (CRD) 上，参考：labels	{}
schedulerName	TiDB 集群使用的调度器	tidb-scheduler
timezone	TiDB 集群默认时区	UTC

参数名	说明	默认值
pvReclaimPolicy	TiDB 集群使用的 PV (Persistent Volume) 的 reclaim policy	Retain
services [0]. name	TiDB 集群对外暴露服务的名字	nil
services [0]. type	TiDB 集群对外暴露服务的类型，(从 ClusterIP、NodePort、LoadBalancer 中选择)	nil

参数名	说明	默认值
discovery.image	TiDB 集群 PD 服务发现组件的镜像，该组件用于在 PD 集群第一次启动时，为各个 PD 实例提供服务发现功能以协调启动顺序	pingcap/tidb-operator:v1.0.0-beta.3
discovery.imagePullPolicy	PD 服务发现组件镜像的拉取策略	IfNotPresent
discovery.resources.limits.cpu	PD 服务发现组件的 CPU 资源限额	250m

参数名	说明	默认值
discovery.resources.limits.memory	PD 服务发现组件的内存资源限额	150Mi
discovery.resources.requests.cpu	PD 服务发现组件的 CPU 资源请求	80m
discovery.resources.requests.memory	PD 服务发现组件的内存资源请求	50Mi

参数名	说明	默认值
enableConfigMapRollout ↪	是否开启 TiDB 集群自动滚动更新。如果启用，则 TiDB 集群的 ConfigMap 变更时，TiDB 集群自动更新对应组件。该配置只在 TiDB Operator v1.0 及以上版本才支持	false

参数名	说明	默认值

pd.config	配置文件格式的 PD 的配置，请参考 pd/conf/config.toml 查看默认 PD 配置文件（选择对应 PD 版本的 tag），可以参考 PD 配置文件描述查看配置参数的具体介绍（请选择对应的文档版本），这里只需要按照配置文件中	TiDB Operator 版本 <= v1.0.0-beta.3，默认值为：
		`[log]`
		`level = "info"`
		`[replication]`
		`location-labels = ["region", "zone", "rack", "host"]`
		配置示例：
		`config: |`
		`  [log]`

参数名	说明	默认值
pd.replicas	PD 的 Pod 数	3
pd.image	PD 镜像	pingcap.com/pd:v3.0.0-rc.1
pd.imagePullPolicy	PD 镜像的拉取策略	IfNotPresent
pd.logLevel	PD 日志级别。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 pd.config 配置：[log] level = "info"	info

参数名	说明	默认值
pd.storageClassName	PD 使用的 storageClassName，指代一种由 Kubernetes 集群提供的存储类型，不同的类可能映射到服务质量级别、备份策略或集群管理员确定的任意策略。详细参考：storage-classes	local-storage

参数名	说明	默认值
pd.maxStoreDownTime	指一个 store 节点断开连接多长时间后状态会被标记为 down，当状态变为 down 后，store 节点开始迁移数据到其它 store 节点如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 pd.config 配置：[schedule] max	pd.maxStoreDownTime	30m

参数名	说明	默认值
pd.maxReplicas	pd.maxReplicas 是 TiDB 集群的数据的副本数。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 pd.config 配置：[replication] max-replicas = 3	3
pd.resources.limits.cpu	每个 PD Pod 的 CPU 资源限额	nil

参数名	说明	默认值
pd.resources.limits.memory	每个 PD Pod 的内存资源限额	nil
pd.resources.limits.storage	每个 PD Pod 的存储容量限额	nil
pd.resources.requests.cpu	每个 PD Pod 的 CPU 资源请求	nil
pd.resources.requests.memory	每个 PD Pod 的内存资源请求	nil
pd.resources.requests.storage	每个 PD Pod 的存储容量请求	1Gi

参数名	说明	默认值
pd.affinity	pd.affinity 定义 PD 的调度规则和偏好，详细请参考：affinity-and-anti-affinity	{}
pd.nodeSelector	pd.nodeSelector 确保 PD Pods 只调度到以该键值对作为标签的节点，详情参考：nodes-elector	{}

参数名	说明	默认值
pd.tolerations	pd.tolerations 应用于 PD Pods，允许 PD Pods 调度到含有指定 taints 的节点上，详情参考：taint-and-toleration	{}
pd.annotations	为 PD Pods 添加特定的 annotations	{}

参数名	说明	默认值
tikv.config	配置文件格式的 TiKV 的配置，请参考 tikv/etc/config-template.toml 查看默认 TiKV 配置文件（选择对应 TiKV 版本的 tag），可以参考 TiKV 配置文件描述查看配置参数的具体介绍（请选择对应的文档版本），这里只需	TiDB Operator 版本 <= v1.0.0-beta.3，默认值为：nil ↪ TiDB Operator 版本 > v1.0.0-beta.3，默认值为：log-level = "info" ↪ "配置示例：config: | log- ↪ level = " ↪ info ↪ "

参数名	说明	默认值
tikv. ↪ replicas	TiKV 的 Pod 数	3
tikv. ↪ image	TiKV 的镜像	pingcap ↪ / ↪ tikv ↪ :v3 ↪ .0.0- ↪ rc ↪ .1
tikv. ↪ imagePullPolicy	TiKV 镜像的拉取策略	IfNotPresent
tikv. ↪ logLevel	TiKV 的日志级别如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tikv. ↪ config ↪ 配置：log- ↪ level ↪ = ↪ " ↪ info ↪ "	info

参数名	说明	默认值
`tikv.storageClassName`	TiKV 使用的 storageClass，storageClassName 指代一种由 Kubernetes 集群提供的存储类型，不同的类可能映射到服务质量级别、备份策略或集群管理员确定的任意策略。详细参考：storage-classes	`local-storage`

参数名	说明	默认值
tikv. ↪ syncLog ↪	syncLog 指是否启用 raft 日志同步功能，启用该功能能保证在断电时数据不丢失。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tikv. ↪ config ↪ 配置： [↪ raftstore ↪] ↪ sync ↪ - ↪ log ↪ = ↪ ↪ true ↪	true

参数名	说明	默认值
tikv. ↪ grpcConcurrency	配置 gRPC server 线程池大小。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tikv. ↪ config ↪ 配置： [↪ server ↪] ↪ grpc ↪ - ↪ concurrency ↪ = ↪ 4	4
tikv. ↪ resources ↪ . ↪ limits ↪ . ↪ cpu	每个 TiKV Pod 的 CPU 资源限额	nil
tikv. ↪ resources ↪ . ↪ limits ↪ . ↪ memory	每个 TiKV Pod 的内存资源限额	nil

参数名	说明	默认值
tikv.resources.limits.storage	每个 TiKV Pod 的存储容量限额	nil
tikv.resources.requests.cpu	每个 TiKV Pod 的 CPU 资源请求	nil
tikv.resources.requests.memory	每个 TiKV Pod 的内存资源请求	nil
tikv.resources.requests.storage	每个 TiKV Pod 的存储容量请求	10Gi

参数名	说明	默认值
tikv.affinity	tikv.affinity 定义 TiKV 的调度规则和偏好，详细请参考：affinity-and-anti-affinity	{}
tikv.nodeSelector	tikv.nodeSelector 确保 TiKV Pods 只调度到以该键值对作为标签的节点，详情参考：nodes-elector	{}

参数名	说明	默认值
tikv.tolerations	tikv.tolerations 应用于 TiKV Pods，允许 TiKV Pods 调度到含有指定 taints 的节点上，详情参考：taint-and-toleration	{}
tikv.annotations	为 TiKV Pods 添加特定的 annotations	{}

参数名	说明	默认值
tikv. ↪ defaultCf. ↪ block ↪ BlockCacheSize	指定 block 缓存 大小， block 缓存 用于 缓存 未压 缩的 block， 较大 的 block 缓存 设置 可以 加快 读取 速度。 一般 推荐 设置 为 tikv. ↪ resources ↪ . ↪ limits ↪ . ↪ memory ↪ 的 30%- 50% 如果 TiDB Oper- ator 版本 > v1.0.0- beta.3， 请通 过 tikv. ↪ config ↪ 配置：	1GB

参数名	说明	默认值
tikv. ↪ writecf.block	指定 writeCF 的 block 缓存大小，一般推荐设置为 tikv. ↪ resources ↪ . ↪ limits ↪ . ↪ memory ↪ 的 10%-30% 如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tikv. ↪ config ↪ 配置：[↪ rocksdb ↪ . ↪ writecf ↪] ↪ block ↪ - ↪ cache ↪ - ↪ size ↪ = ↪ "256 ↪ MB	256MB ↪ BlockCacheSize

参数名	说明	默认值
tikv. ↪ readpool.storage ↪	TiKV 存储的高优先级/普通优先级/低优先级操作的线程池大小如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tikv. ↪ config ↪ 配置： [↪ readpool ↪ . ↪ storage ↪] ↪ high ↪ - ↪ concurrency ↪ = ↪ 4 ↪ normal ↪ - ↪ concurrency ↪ = ↪ 4 ↪ low ↪ - ↪ concurrency ↪ = ↪ 4	4

参数名	说明	默认值
tikv. ↪ readpool.coprocessorConcurrency	一般如果tikv. ↪ resources ↪ . ↪ limits ↪ . ↪ cpu ↪ >8，则tikv. ↪ readpoolCoprocessorConcurrency ↪ 设置为tikv ↪ . ↪ resources ↪ . ↪ limits ↪ . ↪ cpu ↪ *0.8 如果TiDB Operator版本>v1.0.0-beta.3，请通过tikv. ↪ config ↪ 配置：[↪ readpool ↪ . ↪ coprocessor ↪] ↪ high ↪ - ↪ concurrency ↪ = ↪ 8 ↪ normal	8

参数名	说明	默认值
tikv. ↪ storage.scheduler-worker-pool-size	TiKV 调度程序的工作池大小，应在重写情况下增加，同时应小于总 CPU 核心。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tikv. ↪ config ↪ 配置： [↪ storage ↪] ↪ scheduler ↪ - ↪ worker ↪ - ↪ pool ↪ - ↪ size ↪ = ↪ 4	4

参数名	说明	默认值
tidb.config	配置文件格式的 TiDB 的配置，请参考配置文件查看默认 TiDB 配置文件（选择对应 TiDB 版本的 tag），可以参考 TiDB 配置文件描述查看配置参数的具体介绍（请选择对应的文档版本）。这里只需要按照配置文件中的格式修	TiDB Operator 版本 <= v1.0.0-beta.3，默认值为：nil TiDB Operator 版本 > v1.0.0-beta.3，默认值为： [log] level = "info" "配置示例： config: | [log] level = "info" "

参数名	说明	默认值
tidb.replicas	TiDB 的 Pod 数	2
tidb.image	TiDB 的镜像	pingcap/tidb:v3.0.0-rc.1
tidb.imagePullPolicy	TiDB 镜像的拉取策略	IfNotPresent
tidb.logLevel	TiDB 的日志级别。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb.config 配置：[log] level = "info"	info

参数名	说明	默认值
tidb.resources.limits.cpu	每个 TiDB Pod 的 CPU 资源限额	nil
tidb.resources.limits.memory	每个 TiDB Pod 的内存资源限额	nil
tidb.resources.requests.cpu	每个 TiDB Pod 的 CPU 资源请求	nil
tidb.resources.requests.memory	每个 TiDB Pod 的内存资源请求	nil

参数名	说明	默认值
tidb.passwordSecretName	存放 TiDB 用户名及密码的 Secret 的名字，该 Secret 可以使用以下命令创建机密： kubectl create secret generic tidb secret --literal=root=${password}--namespace=${namespace}, 如果没有设置，则 TiDB	nil

参数名	说明	默认值
tidb.initSql	在 TiDB 集群启动成功后，会执行的初始化脚本	nil
tidb.affinity	定义 TiDB 的调度规则和偏好，详细请参考：affinity-and-anti-affinity	{}

参数名	说明	默认值
tidb.nodeSelector	tidb.nodeSelector：确保 TiDB Pods 只调度到以该键值对作为标签的节点，详情参考：nodes-elector	{}
tidb.tolerations	tidb.tolerations：应用于 TiDB Pods，允许 TiDB Pods 调度到含有指定 taints 的节点上，详情参考：taint-and-toleration	{}

参数名	说明	默认值
tidb.annotations	为 TiDB Pods 添加特定的 annotations	{}
tidb.maxFailoverCount	TiDB 最大的故障转移数量，假设为 3 即最多支持同时 3 个 TiDB 实例故障转移	3
tidb.service.type	TiDB 服务对外暴露类型	NodePort

参数名	说明	默认值
tidb. ↪ service. ↪ externalTrafficPolicy	表示此服务是否希望将外部流量路由到节点本地或集群范围的端点。有两个可用选项：Cluster（默认）和 Local。Cluster 隐藏了客户端源 IP，可能导致流量需要二次跳转到另一个节点，但具有良好的整体负载分布。Local	nil

参数名	说明	默认值
tidb. ↪ service. ↪ loadBalancerIP	指定 TiDB 负载均衡器 IP，某些云提供程序允许您指定 loadBalancerIP。在这些情况下，将使用用户指定的 loadBalancerIP 创建负载平衡器。如果未指定 loadBalancerIP 字段，则将使用临时 IP 地址设置 loadBalancer。如果指定 loadBal-	nil

参数名	说明	默认值
tidb.service.	TiDB 服务暴露的 mysqlNodePort	
tidb.service.	TiDB 服务是否暴露 status 端口	true
tidb.service.	指定 TiDB 服务的状态端口暴露的 NodePort	
tidb.separateSlowLog	是否以 sidecar 方式运行独立容器输出 TiDB 的 SlowLog	如果 TiDB Operator 版本 <= v1.0.0-beta.3，默认值为 false。如果 TiDB Operator 版本 > v1.0.0-beta.3，默认值为 true

参数名	说明	默认值
tidb.slowLog.slowLogTailer.image	TiDB 的 slowLogTailer 的镜像，slowLog-Tailer 是一个 sidecar 类型的容器，用于输出 TiDB 的 SlowLog，该配置仅在 tidb.separateSlowLog=true 时生效	busybox:1.26.2
tidb.slowLog.slowLogTailer.resources.limits.cpu	每个 TiDB Pod 的 slowLog-Tailer 的 CPU 资源限额	100m

参数名	说明	默认值
tidb. ↪ slowLogTailer ↪ . ↪ resources ↪ . ↪ limits ↪ . ↪ memory ↪	每个 TiDB slowLogTailer Pod 的 resources slowLog-Tailer 的内存资源限额	50Mi
tidb. ↪ slowLogTailer ↪ . ↪ resources ↪ . ↪ requests ↪ . ↪ cpu	每个 TiDB slowLogTailer Pod 的 resources slowLog-Tailer 的 CPU 资源请求	20m
tidb. ↪ slowLogTailer ↪ . ↪ resources ↪ . ↪ requests ↪ . ↪ memory ↪	每个 TiDB slowLogTailer Pod 的 resources slowLog-Tailer 的内存资源请求	5Mi
tidb. ↪ plugin ↪ . ↪ enable ↪	是否启用 TiDB 插件功能	false
tidb. ↪ plugin ↪ . ↪ directory ↪	指定 TiDB 插件所在的目录	/ ↪ plugins ↪

参数名	说明	默认值
tidb.plugin.list	指定 TiDB 加载的插件列表，plugin ID 命名规则：插件名 -版本，例如：‘conn_limit-1’	[]

参数名	说明	默认值
tidb. ↪ prepared-plan-cache.enabled	是否启用 TiDB 的 prepared plan 缓存。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb. ↪ config ↪ 配置： [↪ prepared ↪ - ↪ plan ↪ - ↪ cache ↪] ↪ enabled ↪ = ↪ ↪ false ↪	false

参数名	说明	默认值
tidb. ↪ preparedPlanCacheCapacity	TiDB 的 pre-pared plan 缓存数量。如果 TiDB Oper-ator 版本 > v1.0.0-beta.3，请通过 tidb. ↪ config ↪ 配置：[↪ prepared ↪ - ↪ plan ↪ - ↪ cache ↪] ↪ capacity ↪ = ↪ ↪ 100 ↪	100

参数名	说明	默认值
tidb. ↪ txnLocalLatchesEnabled ↪	是否启用事务内存锁，当本地事务冲突比较多时建议开启。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb. ↪ config ↪ 配置：[txn- ↪ local ↪ - ↪ latches ↪] ↪ enabled ↪ = ↪ ↪ false ↪	false

参数名	说明	默认值
tidb. ↪ txnLocalLatches ↪	事务内存锁的容量，Hash 对应的 slot 数，会自动向上调整为 2 的指数倍。每个 slot 占 32 Bytes 内存。当写入数据的范围比较广时（如导数据），设置过小会导致变慢，性能下降。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb.	10240000 ↪ Capacity

参数名	说明	默认值
tidb. ↪ token-limit	TiDB 并发执行会话的限制。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb. ↪ config ↪ 配置：token ↪ - ↪ limit ↪ = ↪ ↪ 1000 ↪	1000

参数名	说明	默认值
tidb.↪ memQuotaQuery↪ ↪	TiDB 查询的内存限额，默认 32GB。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb.↪ config ↪ 配置：mem-↪ quota ↪ - ↪ query ↪ = ↪ ↪ 34359738368 ↪	34359738368

参数名	说明	默认值
tidb. ↪ checkMb4ValueInUtf8 ↪	用于控制当字符集为 utf8 时是否检查 mb4 字符。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb. ↪ config ↪ 配置： check ↪ - ↪ mb4 ↪ - ↪ value ↪ - ↪ in ↪ - ↪ utf8 ↪ = ↪ ↪ true ↪	true

参数名	说明	默认值
tidb. ↪ treat_old_version_utf8_as_utf8mb4 ↪	用于升级兼容性。设置为 true 将把旧版本的表/列的 utf8 字符集视为 utf8mb4 字符集如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb. ↪ config ↪ 配置： treat ↪ - ↪ old ↪ - ↪ version ↪ - ↪ utf8 ↪ - ↪ as ↪ - ↪ utf8mb4 ↪ = ↪ ↪ true	true

参数名	说明	默认值
tidb. ↪ lease ↪	tidb. ↪ lease ↪ 是 TiDB Schema lease 的期限，对其更改是非常危险的，除非你明确知道可能产生的结果，否则不建议更改。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb. ↪ config ↪ 配置: lease ↪ = ↪ ↪ "45 ↪ s"	45s

参数名	说明	默认值
tidb.maxProcs	最大可使用的 CPU 核数，0 代表机器/Pod 上的 CPU 数量。如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 tidb.config 配置：[performance] max-procs = 0	0

11.7.3 tidb-backup chart 配置

tidb-backup 是一个用于 Kubernetes 上 TiDB 集群备份和恢复的 Helm Chart。本文详细介绍了 tidb-backup 的可配置参数。

> **注意：**

对于 TiDBOperator v1.1 及以上版本，不再建议使用 tidb-backup chart 部署、管理 TiDB 集群备份，详细信息请参考TiDB Operator v1.1 重要注意事项。

11.7.3.1 mode

- 运行模式
- 默认：“backup”
- 可选值为 backup（备份集群数据）和 restore（恢复集群数据）

11.7.3.2 clusterName

- 目标集群名字
- 默认：“demo”
- 指定要从哪个集群进行备份或将数据恢复到哪个集群中

11.7.3.3 name

- 备份名
- 默认值：“fullbackup-*date*‘{date}‘ 是备份的开始时间，精确到分钟
- 备份名用于区分不同的备份数据

11.7.3.4 secretName

- 访问目标集群时使用的凭据
- 默认：“backup-secret”
- 该 Kubernetes Secret 中需要存储目标集群的登录用户名和密码，你可以通过以下命令来创建这个 Secret：

```
kubectl create secret generic backup-secret -n ${namespace} --from-
    ↪ literal=user=root --from-literal=password=${password}
```

11.7.3.5 storage.className

- Kubernetes StorageClass
- 默认：“local-storage”
- 备份任务需要绑定一个持久卷 (Persistent Volume, PV) 来永久或临时存储备份数据，StorageClass 用于声明持久卷使用的存储类型，需要确保该 StorageClass 在 Kubernetes 集群中存在。

11.7.3.6 storage.size

- 持久卷的空间大小
- 默认：“100Gi”

11.7.3.7 backupOptions

- 备份参数
- 默认：“–chunk-filesize=100”
- 为备份数据时使用的 Mydumper 指定额外的运行参数

11.7.3.8 restoreOptions

- 恢复参数
- 默认：“-t 16”
- 为恢复数据时使用的 Loader 指定额外的运行参数

11.7.3.9 gcp.bucket

- Google Cloud Storage 的 bucket 名字
- 默认：“”
- 用于存储备份数据到 Google Cloud Storage 上

> 注意：
>
> 一旦设置了 gcp 下的任何参数，备份和恢复就会使用 Google Cloud Storage 进行数据存储。也就是说，假如要设置 gcp 下的参数，就需要保证所有 gcp 相关参数都得到正确配置，否则会造成备份恢复失败。

11.7.3.10 gcp.secretName

- 访问 GCP 时使用的凭据
- 默认：“”
- 该 Kubernetes Secret 中需要存储 GCP 的 Service Account 凭据，你可以参考 Google Cloud Documentation 来下载凭据文件，然后通过下面的命令创建 Secret：

```
kubectl create secret generic gcp-backup-secret -n ${namespace} --from-
    ↪ file=./credentials.json
```

11.7.3.11 ceph.endpoint

- Ceph 对象存储的 Endpoint
- 默认：“”
- 用于访问 Ceph 对象存储

> 注意：
>
> 一旦设置了 ceph 下的任何参数，备份和恢复就会使用 Ceph 对象存储进行数据存储。也就是说，假如要设置 ceph 下的参数，就需要保证所有 ceph 相关参数都得到正确配置，否则会造成备份恢复失败。

11.7.3.12 ceph.bucket

- Ceph 对象存储的 bucket 名字
- 默认：“”
- 用于存储数据到 Ceph 对象存储上

11.7.3.13 ceph.secretName

- 访问 Ceph 对象存储时使用的凭据
- 默认：“”
- 该 Kubernetes Secret 中需要存储访问 Ceph 时使用的 access_key 和 secret_key。可使用如下命令来创建这个 Secret：

```
kubectl create secret generic ceph-backup-secret -n ${namespace} --from
    ↪ -literal=access_key=${access_key} --from-literal=secret_key=${
    ↪ secret_key}
```

11.8 日志收集

系统与程序的运行日志对排查问题以及实现一些自动化操作可能非常有用。本文将简要说明收集 TiDB 及相关组件日志的方法。

11.8.1 TiDB 与 Kubernetes 组件运行日志

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。对于 Kubernetes 而言，这些日志会被存放在宿主机的 `/var/log/containers` 目录下，并且文件名中包含了 Pod 和容器名称等信息。因此，可以直接在宿主机上收集容器中应用的日志。

如果在你的现有基础设施中已经有用于收集日志的系统，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，也可以使用你熟悉的任意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：

- 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、分析系统当中。

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日志收集方案可以选择。

11.8.2 系统日志

系统日志可以通过常规方法在 Kubernetes 宿主机上收集，如果在你的现有基础设施中已经有用于收集日志的系统，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，也可以使用你熟悉的任意日志收集系统或方案。

上文提到的几种常见日志收集工具均支持对系统日志的收集，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日志收集方案可以选择。

11.9 Kubernetes 的监控与告警

本文介绍如何对 Kubernetes 集群进行监控及配置告警。

11.9.1 Kubernetes 的监控

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。

11.9.1.1 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，也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，或运行于 Kubernetes 集群内，不同部署方式除在安装配置与资源利用上存在少许差异，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：

- CollectD
- Nagios
- Prometheus & node_exporter
- 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。

我们推荐通过 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。

11.9.1.2 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。

我们推荐通过 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。

11.9.2 Kubernetes 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。

12 版本发布历史

12.1 v1.1

12.1.1 TiDB Operator 1.1.15 Release Notes

发布日期：2022 年 2 月 17 日

TiDB Operator **版本：**1.1.15

12.1.1.1 Bug 修复

- 修复 TiDB Operator 计算 TiKV Region leader 数量时可能会造成 goroutine 泄露的问题 (#4291, [@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.2 TiDB Operator 1.1.14 Release Notes

发布日期：2021 年 10 月 21 日

TiDB Operator **版本：**1.1.14

12.1.2.1 Bug 修复

- 修复 `tidb-backup-manager` 和 `tidb-operator` 镜像中的安全漏洞 (#4217, [@KanShiori](https://github.com/KanShiori))

12.1.3 TiDB Operator 1.1.13 Release Notes

发布日期：2021 年 7 月 2 日

TiDB Operator **版本：**1.1.13

12.1.3.1 优化提升

- 支持为 TiCDC 配置到下游的 TLS 证书 (#3926, [@handlerww](https://github.com/handlerww))
- 支持在未为 BR `toolImage` 指定 tag 时将 TiKV 版本作为 tag (#4048, [@KanShiori](https://github.com/KanShiori))
- 支持在扩缩容 TiDB 过程中协调 PVC (#4033, [@csuzhangxc](https://github.com/csuzhangxc))
- 支持在 backup 日志中隐藏对数据库密码的展示 (#3979, [@dveeden](https://github.com/dveeden))

12.1.3.2 Bug 修复

- 修复部署异构集群时 TiDB Operator 可能 panic 的问题 (#4054 #3965, [@KanShiori](https://github.com/KanShiori) [@july2993](https://github.com/july2993))
- 修复 TiDB 实例缩容后在 TiDB Dashboard 中仍然存在的问题 (#3929, [@july2993](https://github.com/july2993))

12.1.4 TiDB Operator 1.1.12 Release Notes

发布日期：2021 年 4 月 15 日

TiDB Operator 版本：1.1.12

12.1.4.1 新功能

- 支持为备份和恢复 Job 设置自定义环境变量 (#3833，[@dragonly](https://github.com/dragonly))
- 支持备份恢复 CR 设置 affinity 和 tolerations (#3835，[@dragonly](https://github.com/dragonly))
- 设置 `appendReleaseSuffix` 为 `true` 时，支持 tidb-operator chart 使用新的 service account (#3819，[@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.4.2 优化提升

- TiDBInitializer 中增加重试机制，解决 DNS 查询异常处理问题 (#3884，[@handlerww](https://github.com/handlerww))
- 在 PD 的扩缩容和容灾过程中增加多 PVC 支持 (#3820，[@dragonly](https://github.com/dragonly))
- 优化 `PodsAreChanged` 函数 (#3901, [@shonge](https://github.com/shonge))

12.1.4.3 Bug 修复

- 修复 PD/TiKV 挂载多 PVC 时容量设置错误的问题 (#3858，[@dragonly](https://github.com/drago
- 修复创建 `.spec.tidb` 为空并开启 TLS 的 TidbCluster 导致 tidb-controller-manager panic 的问题 (#3852，[@dragonly](https://github.com/dragonly))
- 修复 PD 与 DM 的 `UnjoinedMembers` 中 PVC 状态异常的问题 (#3836, [@dragonly](https://github.com/dragonly))

12.1.5 TiDB Operator 1.1.11 Release Notes

发布日期：2021 年 2 月 26 日

TiDB Operator 版本：1.1.11

12.1.5.1 新功能

- 优化 LeaderElection，支持用户配置 LeaderElection 时间 (#3794, [@july2993](https://github.com/ju
- 支持设置自定义 Store 标签并根据 Node 标签获取值 (#3784, [@L3T](https://github.com/L3T))

12.1.5.2 优化提升

- 添加 TiFlash 滚动更新机制避免升级期间所有 TiFlash Store 同时不可用 (#3789, [@handlerww](https://github.com/handlerww))
- 由从 PD 获取 region leader 数量改为从 TiKV 直接获取以确保拿到准确的 region leader 数量 (#3801, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 打印 RocksDB 和 Raft 日志到 stdout，以支持这些日志的收集和查询 (#3768, [@baurine](https://github.com/baurine))

12.1.6 TiDB Operator 1.1.10 Release Notes

发布日期：2021 年 1 月 28 日

TiDB Operator 版本：1.1.10

12.1.6.1 兼容性改动

- 由于 #3638 的改动，TiDB Operator chart 中创建的 ClusterRoleBinding、ClusterRole、RoleBinding、Role 的 `apiVersion` 从 `rbac.authorization.k8s.io/v1beta1` 更改为 `rbac.authorization.k8s.io/v1`，此时通过 `helm upgrade` 升级 TiDB Operator 可能会报下面错误：

```
Error: UPGRADE FAILED: rendered manifests contain a new resource that
    ↪ already exists. Unable to continue with update: existing resource
    ↪  conflict: namespace: , name: tidb-operator:tidb-controller-
    ↪ manager, existing_kind: rbac.authorization.k8s.io/v1, Kind=
    ↪ ClusterRole, new_kind: rbac.authorization.k8s.io/v1, Kind=
    ↪ ClusterRole
```

详情可以参考 helm/helm#7697，此时需要通过 `helm uninstall` 删除 TiDB Operator 然后重新安装（删除 TiDB Operator 不会影响现有集群）。

12.1.6.2 滚动升级改动

- 由于 #3684 的改动，升级 TiDB Operator 会导致 TidbMonitor Pod 删除重建

12.1.6.3 新功能

- TiDB Operator 灰度升级 (#3548, [@shonge](https://github.com/shonge), #3554, [@cvvz](https://github.com/cvvz))
- TidbMonitor 支持 `remotewrite` (#3679, [@mikechengwei](https://github.com/mikechengwei))
- 支持为 TiDB 集群各组件配置 init containers (#3713, [@handlerww](https://github.com/handlerww)
- TiDB Lightning chart 支持 local backend (#3644, [@csuzhangxc](https://github.com/csuzhangxc))

12.1.6.4 优化提升

- 支持为 TiDB slow log 自定义存储 (#3731, [@BinChenn](https://github.com/BinChenn))
- 为 TidbMonitor 中的 scrape jobs 增加 `tidb_cluster` label 以支持多集群监控 (#3750, [@mikechengwei](https://github.com/mikechengwei))
- TiDB Lightning chart 支持持久化 checkpoint (#3653, [@csuzhangxc](https://github.com/csuzhangxc)
- 将 TidbMonitor 自定义告警规则的存储路径从 `tidb:${tidb_image_version}` 修改为 `tidb:${initializer_image_version}`，确保后续 TiDB 集群升级时不会导致 TidbMonitor Pod 重建 (#3684, [@BinChenn](https://github.com/BinChenn))

12.1.6.5 Bug 修复

- 修复在集群开启 TLS 的情况下，如果不配置 `spec.from` 或者 `spec.to`，使用 BR 备份或者恢复会失败的问题 (#3707, [@BinChenn](https://github.com/BinChenn))
- 修复在开启 Advanced StatefulSet 并且为 PD、TiKV 设置了 `delete-slots` annotations 的情况下，序号大于 `replicas - 1` 的 Pod 在升级过程中不会进行迁移 leader 的问题 (#3702, [@cvvz](https://github.com/cvvz))
- 修复备份或者恢复 Pod 被驱逐或者强制停止的情况下，Backup 或者 Restore 状态没有正常更新为 `Failed` 的问题 (#3696, [@csuzhangxc](https://github.com/csuzhangxc))
- 修复如果 TiKV 集群由于配置错误无法启动，修改 `TidbCluster` CR 配置后，仍然无法启动的问题 (#3694, [@cvvz](https://github.com/cvvz))

12.1.7 TiDB Operator 1.1.9 Release Notes

发布日期：2020 年 12 月 28 日

TiDB Operator 版本：1.1.9

12.1.7.1 优化提升

- 支持使用 `spec.toolImage` 来为 `Backup` 和 `Restore` 指定 Dumpling/TiDB Lightning 的二进制可执行文件 (#3641, [@BinChenn](https://github.com/BinChenn))

12.1.7.2 Bug 修复

- 修复 Prometheus 不能拉取 TiKV Importer 的 metrics (#3631, [@csuzhangxc](https://github.com/cs
- 修复用 BR 和 GCS 进行备份与恢复时的兼容性问题 (#3654, [@dragonly](https://github.com/drago

12.1.8 TiDB Operator 1.1.8 Release Notes

发布日期：2020 年 12 月 21 日

TiDB Operator 版本：1.1.8

12.1.8.1 新功能

- 支持为 PD、TiDB、TiKV、TiFlash、Backup 和 Restore 指定任意 Volume 和 VolumeMount，用户可以利用该功能实现基于 NFS 或者任意 Kubernetes 支持的 Volume 类型的备份和恢复任务。(#3517, [@dragonly](https://github.com/dragonly))

12.1.8.2 优化提升

- 支持在 tidb-lightning 和 tikv-importer helm charts 中为 TiDB 组件和客户端开启 TLS 的功能。(#3598, [@csuzhangxc](https://github.com/csuzhangxc))
- 支持为 TiDB service 指定额外的端口。用户可以利用该功能实现自定义服务，如额外的健康检查机制。(#3599, [@handlerww](https://github.com/handlerww))
- 支持在 TidbInitializer 连接 TiDB server 时不使用 TLS。(#3564, [@LinuxGit](https://github.com/LinuxGit))
- 支持 TiDB Lightning 恢复数据时使用 tableFilters 进行过滤。(#3521, [@sstubbs](https://github.com/sstubbs))
- 支持 Prometheus 从多个 TiDB cluster 抓取 metrics 数据。(#3622, [@mikechengwei](https://github.com/mikechengwei))

  需要手动操作：如果已经部署了 TidbMonitor CR，在升级到 TiDB Operator v1.1.8 之后，需要升级 spec.initializer.version 字段到 v4.0.9，否则有的 metric 在 Grafana 面板上将不会正确显示。Prometheus 抓取 job 名字从 ${component} 改成了 ${namespace}-${TidbCluster Name}-${component}。

- TidbMonitor 中的 Prometheus job 新增了 component label。(#3609, [@mikechengwei](https://github.com/mikechengwei))

12.1.8.3 Bug 修复

- 修复当配置了 spec.tikv.storageVolumes 时，无法部署 TiDB 集群的问题。(#3586, [@mikechengwei](https://github.com/mikechengwei))
- 修复在 TidbInitializer 任务中使用包含非 ASCII 字符密码时的编码错误。(#3569, [@handlerww](https://github.com/handlerww))
- 修复 TiFlash Pods 会被误认为 TiKV Pods 的问题。该问题会导致当同一个 TidbCluster 中部署了 TiFlash 实例时，TiDB Operator 可能将 TiKV 实例缩容到小于 tikv.replicas 的数字。(#3514, [@handlerww](https://github.com/handlerww))
- 修复在开启 TiDB 客户端 TLS 功能的情况下，如果不配置 spec.from，部署 Backup CR 会导致 tidb-controller-manager Pod 崩溃的问题。(#3535, [@dragonly](https://github.com/dragonly))
- 修复了 TiDB Lightning 不打印日志到 stdout 的问题。(#3617, [@csuzhangxc](https://github.com/csu

12.1.9 TiDB Operator 1.1.7 Release Notes

发布日期：2020 年 11 月 13 日

TiDB Operator 版本：1.1.7

12.1.9.1 兼容性变化

- 配置项 `prometheus.spec.config.commandOptions` 的行为发生了变化。该列表中不允许存在重复 flags，否则 Prometheus 无法启动。(#3390, [@mightyguava](https://github.com/mighty
  不能设置的 flags 有：
  - `--web.enable-admin-api`
  - `--web.enable-lifecycle`
  - `--config.file`
  - `--storage.tsdb.path`
  - `--storage.tsdb.retention`

12.1.9.2 新功能

- 新增 `Backup` 和 `Restore` CR 的配置项 `spec.toolImage` 来指定 BR 工具使用的二进制镜像，默认使用 `pingcap/br:${tikv_version}` (#3471, [@namco1992](https://github.com/namco
- 新增配置项 `spec.pd.storageVolumes`、`spec.tidb.storageVolumes` 和 `spec.tikv.storageVolumes`，支持为 PD、TiDB 和 TiKV 挂载多个 PV (#3444 #3425, [@mikechengwei](https://github.com/mikechengwei))
- 新增配置项 `spec.tidb.readinessProbe`，支持使用 `http://127.0.0.0:10080/status` 作为 TiDB 的就绪探针，需要 TiDB 版本 >= v4.0.9 (#3438, [@july2993](https://github.com/july2993))
- PD leader transfer 支持 advanced StatefulSet 启用的情况 (#3395, [@tangwz](https://github.com/tan
- 支持在 TidbCluster CR 中设置 `spec.statefulSetUpdateStrategy` 为 `OnDelete`，来控制 StatefulSets 的更新策略 (#3408, [@cvvz](https://github.com/cvvz))
- 支持故障转移发生时的高可用 (HA) 调度 (#3419, [@cvvz](https://github.com/cvvz))
- 支持将使用 TiDB Ansible 或 TiUP 部署的 TiDB 集群，以及在同一个 Kubernetes 上部署的 TiDB 集群，在线迁移到新的 TiDB 集群 (#3226, [@cvvz](https://github.com/cvvz))
- 在 tidb-scheduler 中支持 advanced StatefulSet (#3388, [@cvvz](https://github.com/cvvz))

12.1.9.3 优化提升

- 当 UP 状态的 stores 数量小于等于 3 时，禁止缩容 TiKV 实例 (#3367, [@cvvz](https://github.com/cvvz))
- 在 `BackupStatus` 和 `RestoreStatus` 中新增 `phase` 状态，用于在 `kubectl get` 返回结果中显示当前最新状态 (#3397, [@namco1992](https://github.com/namco1992))
- BR 在备份和恢复前，跳过设置 `tikv_gc_life_time`，由 BR 自动设置，需要 BR & TiKV 版本 >= v4.0.8 (#3443, [@namco1992](https://github.com/namco1992))

12.1.9.4 Bug 修复

- 修复在当前 `TidbCluster` 之外存在 PD member 的情况下，当前 TiDB Cluster 无法把 PD scale 到 0 的 bug (#3456, [@dragonly](https://github.com/dragonly))

12.1.10 TiDB Operator 1.1.6 Release Notes

发布日期：2020 年 10 月 16 日

TiDB Operator 版本：1.1.6

12.1.10.1 兼容性变化

- 由于 #3342 的改动，`spec.pd.config` 会自动从现有的 YAML 格式转换成 TOML 格式，如果 `spec.pd.config` 中配置了如下参数，升级 TiDB Operator 到 v1.1.6 之后，这个转换无法自动完成，需要编辑 TidbCluster CR 把对应参数的值从 string 格式修改为 bool 格式，例如，从 `"true"` 改为 `true`。
  - `replication.strictly-match-label`
  - `replication.enable-placement-rules`
  - `schedule.disable-raft-learner`
  - `schedule.disable-remove-down-replica`
  - `schedule.disable-replace-offline-replica`
  - `schedule.disable-make-up-replica`
  - `schedule.disable-remove-extra-replica`
  - `schedule.disable-location-replacement`
  - `schedule.disable-namespace-relocation`
  - `schedule.enable-one-way-merge`
  - `schedule.enable-cross-table-merge`
  - `pd-server.use-region-storage`

12.1.10.2 滚动升级改动

- 由于 #3305 的改动，如果 `spec.tidb.annotations` 或者 `spec.tikv.annotations` 中配置了 `tidb.pingcap.com/sysctl-init: "true"`，升级 TiDB Operator 到 v1.1.6 之后 TiDB 或者 TiKV 集群会滚动升级。
- 由于 #3345 的改动，如果集群中部署了 TiFlash，升级 TiDB Operator 到 v1.1.6 之后 TiFlash 集群会滚动升级。

12.1.10.3 新功能

- 添加 `spec.br.options` 支持 Backup 和 Restore CR 自定义 BR 命令行参数 (#3360, [@lichunzhu](https://github.com/lichunzhu))

- 添加 `spec.tikv.evictLeaderTimeout` 支持配置 TiKV evict leader 超时时间 (#3344, [@lichunzhu](https://github.com/lichunzhu))
- 支持使用一个 TidbMonitor 监控多个未开启 TLS 的 TiDB 集群。TidbMonitor CR 添加 `spec.clusterScoped` 配置项，监控多个集群时需要设置为 `true` (#3308, [@mikechengwei](https://github.com/mikechengwei))
- 所有 initcontainers 支持配置资源 (#3305, [@shonge](https://github.com/shonge))
- 支持部署异构 TiDB 集群 (#3003 #3009 #3113 #3155 #3253, [@mikechengwei](https://github.com/mikechengwei))

12.1.10.4 优化提升

- 支持透传 TiFlash 的 TOML 格式配置 (#3355, [@july2993](https://github.com/july2993))
- 支持透传 TiKV/PD 的 TOML 格式配置 (#3342, [@july2993](https://github.com/july2993))
- 支持透传 TiDB 的 TOML 格式配置 (#3327, [@july2993](https://github.com/july2993))
- 支持透传 Pump 的 TOML 格式配置 (#3312, [@july2993](https://github.com/july2993))
- TiFlash proxy 的日志输出到 stdout (#3345, [@lichunzhu](https://github.com/lichunzhu))
- 定时备份到 GCS 时目录名称添加相应备份时间 (#3340, [@lichunzhu](https://github.com/lichunzhu
- 删除 apiserver 和相关的 packages (#3298, [@lonng](https://github.com/lonng))
- 删除 PodRestarter controller 相关的支持 (#3296, [@lonng](https://github.com/lonng))
- 使用 BR metadata 获取备份大小 (#3274, [@lichunzhu](https://github.com/lichunzhu))

12.1.10.5 Bug 修复

- 修复 Discovery 可能导致启动多个 PD 集群的错误 (#3365, [@lichunzhu](https://github.com/lichunz

12.1.11 TiDB Operator 1.1.5 Release Notes

发布日期：2020 年 9 月 18 日

TiDB Operator 版本：1.1.5

12.1.11.1 兼容性变化

- 如果 TiFlash 版本低于 `v4.0.5`，请在将 TiDB Operator 升级到 v1.1.5 和更高版本之前，在 TidbCluster CR 中设置 `spec.tiflash.config.config.flash.service_addr` 为 `{clusterName}-tiflash-POD_NUM.{clusterName}-tiflash-peer.{namespace}.svc:3930` (`{clusterName}` 和 `{namespace}` 需要改为集群实际值)。如果这时需要将 TiFlash 升级到 `v4.0.5` 或更高版本，请同时在 TidbCluster CR 中删除 `spec.tiflash.config.config.flash.service_addr` 项 (#3191, [@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.11.2 新功能

- 支持为 TiDB/Pump/PD 配置 `serviceAccount` (#3246, [@july2993](https://github.com/july2993))
- 支持配置 `spec.tikv.config.log-format` 和 `spec.tikv.config.server.max-grpc-send-msg-len` (#3199, [@kolbe](https://github.com/kolbe))
- 支持配置 TiDB 的 labels 参数 (#3188, [@howardlau1999](https://github.com/howardlau1999))
- 支持从 TiFlash/TiKV 的 failover 中恢复 (#3189, [@DanielZhangQD](https://github.com/DanielZha
- 为 PD/TiKV 添加 `mountClusterClientSecret` 配置，该值为 true 时 Operator 会将 `${cluster_name}-cluster-client-secret` 挂载到 PD/TiKV 容器 (#3282, [@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.11.3 优化提升

- 支持 TiDB/PD/TiKV 的 v4.0.6 配置 (#3180, [@lichunzhu](https://github.com/lichunzhu))
- 挂载集群客户端证书到 PD Pod (#3248, [@weekface](https://github.com/weekface))
- 对于 TiFlash/PD/TiDB，使伸缩实例优先于升级，避免升级失败时无法扩缩容 Pod (#3187, [@lichunzhu](https://github.com/lichunzhu))
- Pump 支持 `imagePullSecrets` 配置 (#3214, [@weekface](https://github.com/weekface))
- 更新 TiFlash 的默认配置项 (#3191, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 移除 TidbMonitor CR 的 ClusterRole 资源 (#3190, [@weekface](https://github.com/weekface))
- 不再重启 Helm 部署的正常退出的 Drainer (#3151, [@lichunzhu](https://github.com/lichunzhu))
- tidb-scheduler 高可用策略将 failover pod 纳入考虑 (#3171, [@cofyc](https://github.com/cofyc))

12.1.11.4 Bug 修复

- 修复 TidbMonitor CR 中的 Grafana container 忽略 `Env` 配置的问题 (#3237, [@tirsen](https://github.com/tirsen))

12.1.12 TiDB Operator 1.1.4 Release Notes

发布日期：2020 年 8 月 21 日

TiDB Operator **版本：**1.1.4

12.1.12.1 重大变化

- 将 `TableFilter` 添加到 `BackupSpec` 和 `RestoreSpec`。`TableFilter` 支持使用 Dumpling 或 BR 备份特定的数据库或表，并支持使用 BR 恢复特定的数据库或表。从 v1.1.4 开始，已弃用 `BackupSpec.Dumpling.TableFilter`，请改为配置 `BackupSpec.TableFilter`。从 TiDB v4.0.3 开始，可以通过配置 `BackupSpec.TableFilter` 来替换 `BackupSpec.BR.DB` 和 `BackupSpec.BR.Table` 字段，并且通过配置 `RestoreSpec.TableFilter` 来替换 `RestoreSpec.BR.DB` 和 `RestoreSpec.BR.Table` 字段 (#3134，[@sstubbs](https://github.com/sstubbs))

- 更新 TiDB 和配套工具的版本为 v4.0.4 (#3135,[@lichunzhu](https://github.com/lichunzhu))
- 支持在 TidbMonitor CR 中为初始化容器自定义环境变量 (#3109,[@kolbe](https://github.com/kolb
- 支持增加存储请求 (#3096, [@cofyc](https://github.com/cofyc))
- 为使用 Dumpling 和 TiDB Lightning 进行备份恢复添加 TLS 支持 (#3100, [@lichunzhu](https://github.com/lichunzhu))
- 支持 TiFlash 中的 `cert-allowed-cn` 配置项 (#3101,[@DanielZhangQD](https://github.com/Daniel
- 在 TidbCluster CRD 中添加对 TiDB 配置项 `max-index-length` 的支持 (#3076, [@kolbe](https://github.com/kolbe))
- 修复了启用 TLS 时的 goroutine 泄漏 (#3081,[@DanielZhangQD](https://github.com/DanielZhangQ
- 修复了启用 TLS 时由 etcd 客户端引起的内存泄漏问题 (#3064, [@DanielZhangQD](https://github.c
- 为 TiFlash 添加 TLS 支持 (#3049,[@DanielZhangQD](https://github.com/DanielZhangQD))
- 为 tidb-operator chart 中部署的 Admission Webhook 和增强型 Statefulset 控制器配置 TZ 环境 (#3034, [@cofyc](https://github.com/cofyc))

12.1.13 TiDB Operator 1.1.3 Release Notes

发布日期：2020 年 7 月 27 日

TiDB Operator **版本：**1.1.3

12.1.13.1 需要采取的行动

- 在 `BackupSpec` 中添加字段 `cleanPolicy`，表示从集群中删除 `Backup` CR 时对备份数据采取的清理策略（默认为 `Retain`）。需要注意的是，在 v1.1.3 之前的版本中，TiDB Operator 将在删除 `Backup` CR 时清除远程存储中的备份数据。因此，如果想像以前一样清除备份数据，请在 `Backup` CR 的 `spec.cleanPolicy` 或 `BackupSchedule` CR 中的 `spec.backupTemplate.cleanPolicy` 设置为 `Delete` (#3002, [@lichunzhu](https://github.com/lichunzhu))

- 将 `mydumper` 替换为 `dumpling` 进行备份。

  如果在 `Backup` CR 中配置了 `spec.mydumper`，或者在 `BackupSchedule` CR 中配置了 `spec.backupTemplate.mydumper`，需要将该配置迁移至 `spec.dumpling` 或 `spec.backupTemplate.dumpling`。请注意，TiDB Operator 升级到 v1.1.3 后，`spec.mydumper` 或 `spec.backupTemplate.mydumper` 配置会丢失 (#2870, [@lichunzhu](https://github.com/lichunzhu))

12.1.13.2 其他需要注意的变更

- 将 backup manager 中的工具更新到 v4.0.3 (#3019,[@lichunzhu](https://github.com/lichunzhu))
- 在 `Backup` CR 中添加 `cleanPolicy` 字段，表示从集群中删除 `Backup` CR 时对远端存储的备份数据采取的清理策略 (#3002, [@lichunzhu](https://github.com/lichunzhu))
- 为 TiCDC 添加 TLS 支持 (#3011, [@weekface](https://github.com/weekface))
- 在 Drainer 和下游数据库服务器之间添加 TLS 支持 (#2993,[@lichunzhu](https://github.com/lichun

- 支持为 TiDB Service 指定 `mysqlNodePort` 和 `statusNodePort` (#2941，[@lichunzhu](https://github.com/lichunzhu))
- 修复 Drainer `values.yaml` 文件中的 `initialCommitTs` 错误 (#2857，[@weekface](https://github.com/weekface))
- 为 TiKV 添加 `backup` 配置，为 PD 添加 `enable-telemetry` 并弃用 `disable-telemetry` 配置 (#2964，[@lichunzhu](https://github.com/lichunzhu))
- 在 `get restore` 命令中添加 commitTS 列 (#2926,[@lichunzhu](https://github.com/lichunzhu))
- 将 Grafana 版本从 v6.0.1 更新到 v6.1.6 (#2923,[@lichunzhu](https://github.com/lichunzhu))
- 在 `Restore` CR 中的 `Status` 字段下增加 commitTS 字段 (#2899，[@lichunzhu](https://github.com/lichunzhu))
- 如果用户尝试清除的备份数据不存在，则不报错并退出 (#2916，[@lichunzhu](https://github.com/lichunzhu))
- 支持在 `TidbClusterAutoScaler` 中设置 TiKV 根据剩余存储容量自动扩容 (#2884，[@Yisaer](https://github.com/Yisaer))
- 清理 `Dumpling` 备份 Job 中的临时文件，以节省空间 (#2897，[@lichunzhu](https://github.com/lichunzhu))
- 如果现有 PVC 的大小小于 `Backup` Job 中的存储请求，则 `Backup` Job 失败 (#2894，[@lichunzhu](https://github.com/lichunzhu))
- 支持在 TiKV store 升级失败时扩缩容和自动故障转移 (#2886,[@cofyc](https://github.com/cofyc))
- 修复了无法设置 `TidbMonitor` 资源的问题 (#2878,[@weekface](https://github.com/weekface))
- 修复了 tidb-cluster chart 中监控组件创建失败的问题 (#2869,[@8398a7](https://github.com/8398a7)
- 在 `TidbClusterAutoScaler` 中删除 `readyToScaleThresholdSeconds`；TiDB Operator 不支持 `TidbClusterAutoScaler` 中的降噪 (#2862,[@Yisaer](https://github.com/Yisaer))
- 将 tidb-backup-manager 中使用的 TiDB Lightning 版本从 v3.0.15 更新到 v4.0.2 (#2865，[@lichunzhu](https://github.com/lichunzhu))

12.1.14 TiDB Operator 1.1.2 Release Notes

发布日期：2020 年 7 月 1 日

TiDB Operator 版本：1.1.2

12.1.14.1 需要采取的行动

- 修复了与 PD 4.0.2 不兼容的问题。在部署 TiDB 4.0.2 及更高版本之前，请将 TiDB Operator 升级到 v1.1.2 (#2809，[@cofyc](https://github.com/cofyc))

12.1.14.2 其他需要注意的变更

- 抓取 TiCDC, TiDB Lightning 和 TiKV Importer 的监控指标 (#2835，[@weekface](https://github.com/weekface))
- 更新 PD/TiDB/TiKV 配置为 v4.0.2 (#2828,[@DanielZhangQD](https://github.com/DanielZhangQ)
- 修复了缩容后 PD 成员可能仍然存在的错误 (#2793,[@Yisaer](https://github.com/Yisaer))

- 如果存在 TidbClusterAutoScaler CR，同步 TidbClusterAutoScaler 信息到 TidbCluster Status 字段 (#2791，[@Yisaer](https://github.com/Yisaer))
- 支持在 TiDB 参数中配置容器生命周期 hook 和 terminationGracePeriodSeconds (#2810，[@weekface](https://github.com/weekface))

12.1.15 TiDB Operator 1.1.1 Release Notes

发布日期：2020 年 6 月 19 日

TiDB Operator 版本：1.1.1

12.1.15.1 重大变化

- 添加 additionalContainers 和 additionalVolumes 字段，以便 TiDB Operator 添加 sidecar 到 TiDB、TiKV、PD 等 (#2229, [@yeya24](https://github.com/yeya24))
- 添加交叉检查以确保 TiKV 不会同时被扩缩容和升级 (#2705, [@DanielZhangQD](https://github.co
- 修复了当未设置 ClusterRef 中的 namespace 时，TidbMonitor 会在不同的 namespace 中抓取多个具有相同名字的 TidbCluster 的监控数据的问题 (#2746, [@Yisaer](https://github.com/Yisaer))
- 更新 TiDB Operator 部署 TiDB Cluster 4.0.0 版本镜像的示例 (#2600, [@kolbe](https://github.com/
- 为 TidbMonitor 添加 alertMangerAlertVersion 配置 (#2744, [@weekface](https://github.com/wee
- 修复滚动升级后监控的告警规则丢失的问题 (#2715, [@weekface](https://github.com/weekface))
- 修复了先缩容再扩容时 Pod 可能长时间停留在 Pending 状态的问题 (#2709, [@cofyc](https://github.com/cofyc))
- 在 PDSpec 中添加 EnableDashboardInternalProxy 配置以使用户可以直接访问 PD Dashboard (#2713, [@Yisaer](https://github.com/Yisaer))
- 修复了当 TidbMonitor 和 TidbCluster 的 reclaimPolicy 值不同时，PV 同步错误的问题 (#2707, [@Yisaer](https://github.com/Yisaer))
- 更新 TiDB Operator 的配置版本到 v4.0.1 (#2702, [@Yisaer](https://github.com/Yisaer))
- 将 tidb-discovery 策略类型更改为 Recreate，以修复可能同时存在多个 discovery pod 的问题 (#2701, [@weekface](https://github.com/weekface))
- 支持在集群开启 TLS 时暴露 Dashboard 服务 (#2684, [@Yisaer](https://github.com/Yisaer))
- 添加 .tikv.dataSubDir 字段以指定数据卷下子目录来存储 TiKV 数据 (#2682, [@cofyc](https://github.com/cofyc))
- 支持向所有组件添加 imagePullSecrets 属性 (#2679, [@weekface](https://github.com/weekface))
- 支持 StatefulSet 和 Pod 验证 webhook 同时工作 (#2664, [@Yisaer](https://github.com/Yisaer))
- 在 TiDB Operator 同步标签到 TiKV stores 失败时提交一个事件 (#2587, [@PengJi](https://github.com/PengJi))
- 在 Backup 和 Restore jobs 日志中隐藏 datasource 信息 (#2652, [@Yisaer](https://github.com/Yisa
- 支持 TidbCluster Spec 配置 enableDynamicConfiguration (#2539, [@Yisaer](https://github.com/Y
- 在 TidbCluster 和 TidbMonitor 的 ServiceSpec 支持 LoadBalancerSourceRanges 配置 (#2610, [@shonge](https://github.com/shonge))
- 当部署了 TidbMonitor 时，支持 TidbCluster 的 Dashboard metrics 功能 (#2483, [@Yisaer](https://github.com/Yisaer))
- 更新 TiDB Operator 中的 DM 的版本到 v2.0.0-beta.1 (#2615, [@tennix](https://github.com/tennix)

- 支持配置 discovery 资源 (#2434, [@shonge](https://github.com/shonge))
- 支持 TidbCluster 自动扩缩容的降噪 (#2307, [@vincent178](https://github.com/vincent178))
- 支持在 TidbMonitor 中抓取 Pump 和 Drainer 监控指标 (#2750, [@Yisaer](https://github.com/Yisae

12.1.16 TiDB Operator 1.1 GA Release Notes

发布日期：2020 年 5 月 28 日

TiDB Operator 版本：1.1.0

12.1.16.1 从 v1.0.x 升级

对于 v1.0.x 的用户，请参考升级 TiDB Operator 来升级集群中的 TiDB Operator。注意在升级之前应该阅读发布说明 (特别是重大变更和需要操作的项目)。

12.1.16.2 v1.0.0 后的重大变更

- 将 TiDB Pod 的 readiness 探针从 HTTPGet 更改为 TCPSocket 4000 端口。这将触发 tidb-server 组件滚动升级。你可以在升级 TiDB Operator 之前将 spec.paused 设置为 true 以避免滚动升级，并在准备升级 tidb-server 时将其重新设置为 false (#2139, [@weekface](https://github.com/weekface))
- 为 tidb-server 配置了 --advertise-address，这将触发 tidb-server 滚动升级。你可以在升级 TiDB Operator 之前将 spec.paused 设置为 true 以避免滚动升级，并在准备升级 tidb-server 时将其设置为 false (#2076, [@cofyc](https://github.com/cofyc))
- --default-storage-class-name 和 --default-backup-storage-class-name 标记被废弃，现在存储类型默认为 Kubernetes 默认存储类型。如果你已经设置的默认存储类型不是 Kubernetes 的默认存储类型，请在 TiDB 集群的 Helm 或 YAML 文件中显式设置它们 (#1581, [@cofyc](https://github.com/cofyc))
- 为所有 Helm chart 添加时区选项 (#1122, [@weekface](https://github.com/weekface))

  对于 tidb-cluster chart，我们已经支持 timezone 选项 (默认为 UTC)。如果用户没有将其改为其他值 (例如 Asia/Shanghai)，则不会重新创建任何 Pod。

  如果用户将其改为其他值 (例如 Asia/Shanghai)，则将重新创建所有相关的 Pod (添加一个 TZ 环境变量)，即滚动更新。

  相关的 Pod 包括 pump、drainer、discovery、monitor、scheduled backup、tidb-initializer 和 tikv-importer。

  TiDB Operator 维护的所有镜像的时区均为 UTC。如果你使用自己的镜像文件，需要保证镜像内的时区为 UTC。

12.1.16.3 其他重要更新

- 修复了同时启用 `PodWebhook` 和增强型 StatefulSet 时 `TidbCluster` 升级的错误 (#2507, [@Yisaer](https://github.com/Yisaer))
- `tidb-scheduler` 中支持 preemption (#2510, [@cofyc](https://github.com/cofyc))
- 将 BR 更新为 v4.0.0-rc.2，包含 `auto_random` 的修复 (#2508, [@DanielZhangQD](https://github.com
- 增强型 StatefulSet 支持 TiFlash (#2469, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 在运行 TiDB 前同步 Pump (#2515, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 删除 `TidbControl` 锁以提高性能 (#2489, [@weekface](https://github.com/weekface))
- 在 `TidbCluster` 中支持 TiCDC 功能 (#2362, [@weekface](https://github.com/weekface))
- 将 TiDB/TiKV/PD 配置更新为 4.0.0 GA 版本 (#2571, [@Yisaer](https://github.com/Yisaer))
- TiDB Operator 将不会对 Failover 创建的 PD Pod 再次进行 Failover (#2570, [@Yisaer](https://github.com/Yisaer))

12.1.17 TiDB Operator 1.1 RC.4 Release Notes

发布日期：2020 年 5 月 15 日

TiDB Operator 版本：1.1.0-rc.4

12.1.17.1 需要采取的行动

- 每个组件可以使用单独的 TiDB 客户端证书。用户应该将 `Backup` 和 `Restore` CR 中的旧的 TLS 配置迁移到新的配置。更多详细信息，请参考#2403(#2403，[@weekface](https://github.com/weekface))

12.1.17.2 其他值得注意的变化

- 修复了 `pingcap.com/last-applied-configuration` 的标注被同步到 `tc.spec.tidb.service.annotations` 的问题 (#2471，[@Yisaer](https:// github.com/Yisaer))
- 修复了 Kubernetes `healthCheckNodePort` 在 TiDB Service 调和过程中一直被修改的问题 (#2438，[@aylei](https://github.com/aylei))
- 在 `TidbCluster` 状态中添加了 `TidbMonitorRef` (#2424，[@Yisaer](https://github.com/Yisaer))
- 支持设置远程存储的备份路径前缀 (#2435，[@onlymellb](https://github.com/onlymellb))
- 在 CRD 备份中支持自定义的 mydumper 选项 (#2407，[@onlymellb](https://github.com/onlymellb)
- 在 `TidbCluster` CR 中支持 TiCDC (#2338，[@weekface](https://github.com/weekface))
- 将 `tidb-backup-manager` 镜像中的 BR 的版本更新为 v3.1.1 (#2425，[@DanielZhangQD](https://gi
- 支持为 ACK 上的 `TiFlash` 和 `CDC` 创建节点池 (#2420，[@DanielZhangQD](https://github.com/Dan
- 支持在 EKS 上为 `TiFlash` 和 `CDC` 创建节点池 (#2413，[@DanielZhangQD](https://github.com/Dan))
- 启用存储时，为 `TidbMonitor` 暴露 `PVReclaimPolicy` (#2379，[@Yisaer](https://github.com/Yisaer)
- 在 tidb-scheduler 中支持任意基于拓扑的 HA (例如节点区域) (#2366，[@PengJi](https://github.co
- 如果 TiDB 版本低于 4.0.0，跳过 PD dashboard 的 TLS 设置 (#2389，[@weekface](https://github.com/weekface))

- 支持使用 BR 对 GCS 进行备份和还原 (#2267,[@shuijing198799](https://github.com/shuijing1987
- 更新 TiDBConfig 和 TiKVConfig 以支持 4.0.0-rc 版本 (#2322,[@Yisaer](https://github.com/Yisa
- 修复了 TidbCluster 中服务类型为 NodePort 时，NodePort 的值会频繁更改的问题 (#2284，[@Yisaer](https://github.com/Yisaer))
- 为 TidbClusterAutoScaler 添加了外部策略功能 (#2279,[@Yisaer](https://github.com/Yisaer))
- 删除 TidbMonitor 时，不会删除 PVC (#2374，[@Yisaer](https://github.com/Yisaer))
- 支持为 TiFlash 扩缩容 (#2237,[@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.18 TiDB Operator 1.1 RC.3 Release Notes

Release date: April 30, 2020

TiDB Operator version: 1.1.0-rc.3

12.1.18.1 Notable Changes

- Skip auto-failover when pods are not scheduled and perform recovery operation no matter what state failover pods are in (#2263, [@cofyc](https://github.com/cofyc))
- Support `TiFlash` metrics in `TidbMonitor` (#2341, [@Yisaer](https://github.com/Yisaer))
- Do not print `rclone` config in the Pod logs (#2343, [@DanielZhangQD](https://github.com/DanielZh
- Using `Patch` in `periodicity` controller to avoid updating `StatefulSet` to the wrong state (#2332, [@Yisaer](https://github.com/Yisaer))
- Set `enable-placement-rules` to `true` for PD if TiFlash is enabled in the cluster (#2328, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Support `rclone` options in the Backup and Restore CR (#2318, [@DanielZhangQD](https://github.co
- Fix the issue that statefulsets are updated during each sync even if no changes are made to the config (#2308, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Support configuring `Ingress` in `TidbMonitor` (#2314, [@Yisaer](https://github.com/Yisaer))
- Fix a bug that auto-created failover pods can't be deleted when they are in the failed state (#2300, [@cofyc](https://github.com/cofyc))
- Add useful `Event` in `TidbCluster` during upgrading and scaling when `admissionWebhook` ↪ `.validation.pods` in operator configuration is enabled (#2305, [@Yisaer](https://github.com/Yi
- Fix the issue that services are updated during each sync even if no changes are made to the service configuration (#2299, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Fix a bug that would cause panic in statefulset webhook when the update strategy of `StatefulSet` is not `RollingUpdate` (#2291, [@Yisaer](https://github.com/Yisaer))
- Fix a panic in syncing `TidbClusterAutoScaler` status when the target `TidbCluster` does not exist (#2289, [@Yisaer](https://github.com/Yisaer))
- Fix the `pdapi` cache issue while the cluster TLS is enabled (#2275, [@weekface](https://github.com/weekface))
- Fix the config error in restore (#2250, [@Yisaer](https://github.com/Yisaer))
- Support failover for TiFlash (#2249, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Update the default `eks` version in terraform scripts to 1.15 (#2238, [@Yisaer](https://github.com/Yis
- Support upgrading for TiFlash (#2246, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Add `stderr` logs from BR to the backup-manager logs (#2213, [@DanielZhangQD](https://github.co

- Add field `TiKVEncryptionConfig` in `TiKVConfig`, which defines how to encrypt data key and raw data in TiKV, and how to back up and restore the master key. See the description for details in `tikv_config.go` (#2151, [@shuijing198799](https://github.com/shuijing198799))

12.1.19 TiDB Operator 1.1 RC.2 Release Notes

Release date: April 15, 2020

TiDB Operator version: 1.1.0-rc.2

12.1.19.1 Action Required

- Change TiDB pod `readiness` probe from `HTTPGet` to `TCPSocket` 4000 port. This will trigger rolling-upgrade for the `tidb-server` component. You can set `spec.paused` ↪ to `true` before upgrading tidb-operator to avoid the rolling upgrade, and set it back to `false` when you are ready to upgrade your TiDB server (#2139, [@weekface](https://github.com/weekface))

12.1.19.2 Notable Changes

- Add `status` field for `TidbAutoScaler` CR (#2182, [@Yisaer](https://github.com/Yisaer))
- Add `spec.pd.maxFailoverCount` field to limit max failover replicas for PD (#2184, [@cofyc](https://github.com/cofyc))
- Emit more events for `TidbCluster` and `TidbClusterAutoScaler` to help users know TiDB running status (#2150, [@Yisaer](https://github.com/Yisaer))
- Add the `AGE` column to show creation timestamp for all CRDs (#2168, [@cofyc](https://github.com/cofyc))
- Add a switch to skip PD Dashboard TLS configuration (#2143, [@weekface](https://github.com/week
- Support deploying TiFlash with TidbCluster CR (#2157, [@DanielZhangQD](https://github.com/Da
- Add TLS support for TiKV metrics API (#2137, [@weekface](https://github.com/weekface))
- Set PD DashboardConfig when TLS between the MySQL client and TiDB server is enabled (#2085, [@weekface](https://github.com/weekface))
- Remove unnecessary informer caches to reduce the memory footprint of tidb-controller-manager (#1504, [@aylei](https://github.com/aylei))
- Fix the failure that Helm cannot load the kubeconfig file when deleting the tidb-operator release during `terraform destroy` (#2148, [@DanielZhangQD](https://github.com/Daniel
- Support configuring the Webhook TLS setting by loading a secret (#2135, [@Yisaer](https://github.com/Yisaer))
- Support TiFlash in TidbCluster CR (#2122, [@DanielZhangQD](https://github.com/DanielZhangQ
- Fix the error that alertmanager couldn't be set in `TidbMonitor` (#2108, [@Yisaer](https://github.com

12.1.20 TiDB Operator 1.1 RC.1 Release Notes

Release date: April 1, 2020

TiDB Operator version: 1.1.0-rc.1

12.1.20.1 需要采取的行动

- 将为 tidb-server 配置 `--advertise-address` 选项。这会触发 tidb-server 组件的滚动升级。您可以在升级 tidb-operator 之前将 `spec.paused` 设置为 `true` 以避免滚动升级的行为，并在准备好升级 tidb-server 版本时将其设置回 `false` (#2076, [@cofyc](https://github.com/cofyc))
- 在 backup and restore spec 中添加 `tlsClient.tlsSecret` 字段。可以通过该字段指定包含证书的密钥的名称 (#2003, [@shuijing198799](https://github.com/shuijing198799))
- 为 `Backup`, `Restore` 以及 `BakcupSchedule` 移除 `spec.br.pd`, `spec.br.ca` ↪ , `spec.br.cert`, `spec.br.key` 选项，添加 `spec.br.cluster`, `spec.br` ↪ `.clusterNamespace` 选项，让 BR 的配置项更加合理 (#1836, [@shuijing198799](https://github.com/shuijing198799))

12.1.20.2 其他需要注意的变更

- 在 `Restore` 中使用 `tidb-lightning` 替代 `loader` (#2068, [@Yisaer](https://github.com/Yisaer))
- 为 TiDB 组件添加 `cert-allowed-cn` 支持 (#2061, [@weekface](https://github.com/weekface))
- 修复 PD `location-labels` 配置项的问题 (#1941, [@aylei](https://github.com/aylei))
- 可以通过 `spec.paused` 控制 TiDB 集群暂停部署 (#2013, [@cofyc](https://github.com/cofyc))
- 在使用 CR 部署 TiDB 集群时，TiDB 的 `max-backups` 配置项默认值设为 `3` (#2045, [@Yisaer](https://github.com/Yisaer))
- 支持为组件配置自定义环境变量 (#2052, [@cofyc](https://github.com/cofyc))
- 修复 `kubectl get tc` 不能正确显示镜像的问题 (#2031, [@Yisaer](https://github.com/Yisaer))
- 在 `spec.tikv.maxFailoverCount` 及 `spec.tidb.maxFailoverCount` 未定义时，将其默认值设为 `3` (#2015, [@Yisaer](https://github.com/Yisaer))
- 在 `maxFailoverCount` 设为 `0` 时禁用自动故障转移的功能 (#2015, [@Yisaer](https://github.com/Yis
- 支持通过 Terraform 使用 TidbCluster 及 TidbMonitor CR 在 ACK 上部署 TiDB 集群 (#2012, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 将 TidbCluster 中的 PDConfig 升级到 PD v3.1.0 (#1928, [@Yisaer](https://github.com/Yisaer))
- 支持通过 Terraform 使用 TidbCluster 及 TidbMonitor CR 在 AWS 上部署 TiDB 集群 (#2004, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 将 TidbCluster 中的 TidbConfig 升级到 TiDB v3.1.0 (#1906, [@Yisaer](https://github.com/Yisaer))
- 允许用户在 TiDB 初始化时为 initContainers 指定资源 (#1938, [@tfulcrand](https://github.com/tfu
- 为 Pump 及 Drainer 添加 TLS 支持 (#1979, [@weekface](https://github.com/weekface))
- 为 auto-scaler 和 initializer 添加文档与示例 (#1772, [@Yisaer](https://github.com/Yisaer))
  - 添加检查以保证当 TidbMonitor 的 serviceType 为 NodePort 时，NodePort 不会被改变
  - 添加 EnvVar 排序来避免控制器从同一份 TidbMonitor 规范渲染出不同的结果

  - 修复 TidbMonitor LoadBalancer IP 不被使用的问题 (#1962, [@Yisaer](https://github.com/Yis

- tidb-initializer 支持 TLS (#1931, [@weekface](https://github.com/weekface))
  - 修复 Advanced StatefulSet 不能与 webhook 工作的问题
  - 把 Down State TiKV pod 在 webhook 中处理删除请求的响应从允许改为拒绝 (#1963, [@Yisaer](https://github.com/Yisaer))
- 修复指定 drainerName 时 drainer 的安装错误 (#1961, [@DanielZhangQD](https://github.com/Dani
- 修正一些 TiKV toml 配置文件中的配置名 (#1887, [@aylei](https://github.com/aylei))
- 支持使用远程目录作为 tidb-lightning 的数据源 (#1629, [@aylei](https://github.com/aylei))
- 添加 API 文档以及生成该文档的脚本 (#1945, [@Yisaer](https://github.com/Yisaer))
- 添加 tikv-importer chart (#1910, [@shonge](https://github.com/shonge))
- 修复当开启 TLS 时 Prometheus 的 scrape 配置问题 (#1919, [@weekface](https://github.com/weekfa
- 为 TiDB 组件间的通信开启 TLS (#1870, [@weekface](https://github.com/weekface))
- 修复在 TiKV 升级过程中当 `Values.admission.validation.pods` 设为 true 时的超时错误 (#1875, [@Yisaer](https://github.com/Yisaer))
- 为 MySQL 客户端的通信开启 TLS (#1878, [@weekface](https://github.com/weekface))
- 修复 TiDB 默认配置设置错误的问题 (#1860, [@Yisaer](https://github.com/Yisaer))
- 如果 targetRef 没定义则使用 TidbMonitor 的 namespace 作为 targetRef (#1834, [@Yisaer](https://github.com/Yisaer))
- 支持使用 `--advertise-address` 参数启动 tidb-server (#1859, [@LinuxGit](https://github.com/Lin
- Backup/Restore: 支持配置 TiKV 的 GC 生命周期 (#1835, [@LinuxGit](https://github.com/LinuxG
- 支持使用 OIDC 对 S3 进行访问鉴权 (#1817, [@tirsen](https://github.com/tirsen))
  - 把之前的配置 `admission.hookEnabled.pods` 改为 `admission.validation.pods`
  - 把之前的配置 `admission.hookEnabled.statefulSets` 改为 `admission.validation.statefulSets`
  - 把之前的配置 `admission.hookEnabled.validating` 改为 `admission.validation.pingcapResources`
  - 把之前的配置 `admission.hookEnabled.defaulting` 改为 `admission.mutation.pingcapResources`
  - 把之前的配置 `admission.failurePolicy.defaulting` 改为 `admission.failurePolicy.mutation`
  - 把之前的配置 `admission.failurePolicy.*` 改为 `admission.failurePolicy.validation`(#1832, [@Yisaer](https://github.com/Yisaer))
- 默认开启 TidbCluster 的 defaulting mutation，当使用 admission webhook 时推荐开启该开关 (#1816, [@Yisaer](https://github.com/Yisaer))
- 修复当集群开启 TLS 的情况下使用 CR 创建集群时 TiKV 启动失败的错误 (#1808, [@weekface](https://github.com/weekface))
- 支持在备份与恢复时在远程存储中使用前缀 (#1790, [@DanielZhangQD](https://github.com/Danie

12.1.21 TiDB Operator 1.1 Beta.2 Release Notes

发布日期：2020 年 2 月 26 日

TiDB Operator 版本：1.1.0-beta.2

12.1.21.1 需要采取的行动

- `--default-storage-class-name` 和 `--default-backup-storage-class-name` 已经废弃，现在 storage class 默认使用 Kubernetes default storage class。如果你曾经将 default storage class 设置为 Kubernetes default storage class 之外的选项，请在 TiDB 集群的 `helm` 或者 YAML 文件中显式设定。(#1581, [@cofyc](https://github.com/cofyc))

12.1.21.2 其他值得注意的改变

- 允许用户为备份和恢复设置亲和性和容忍度 (#1737, [@Smana](https://github.com/Smana))
- 解决 `AdvancedStatefulSet` 和 `Admission Webhook` 一起使用存在的问题 (#1640, [@Yisaer](https://github.com/Yisaer))
- 增加使用 TidbCluster CR 部署 TiDB 集群的样例 (#1573, [@aylei](https://github.com/aylei))
- 支持基于 CPU 平均负载的集群自动扩容特性 (#1731, [@Yisaer](https://github.com/Yisaer))
- 支持数据库与客户端之间用户自定义证书 (#1714, [@weekface](https://github.com/weekface))
- 为 `tidb-backup` 增加一个可以重用已有的 PVC 来恢复集群的选项 (#1708, [@mightyguava](https://github.com/mightyguava))
- 为 `tidb-backup` 增加 `resources`，`imagePullPolicy` 和 `nodeSelector` 字段 (#1705, [@mightyguava](https://github.com/mightyguava))
- 为 TiDB server 的证书增加更多的 SANs（Subject Alternative Name）(#1702, [@weekface](https://github.com/weekface))
- 当 `AdvancedStatfulSet` 开启时，支持自动迁移已有的 `Kubernetes StatefulSets` 到 `Advanced StatefulSets` (#1580, [@cofyc](https://github.com/cofyc))
- 修复了 `admission webhook` 导致删除 PD pod 失败的问题，允许在 PVC 找不到的情况下，删除 PD 和 TiKV pod (#1568, [@Yisaer](https://github.com/Yisaer))
- 限制 PD 和 TiKV 的重启频率，同时只允许一个实例进行重启 (#1532, [@Yisaer](https://github.com/Yisaer))
- 为 `TidbMonitor` 增加默认的与它部署相同的 `ClusterRef` 命名空间，并且修复当 `Spec.PrometheusSpec.logLevel` 丢失的时候，`TidbMonitor` 的 pod 不能被创建出来的问题 (#1500, [@Yisaer](https://github.com/Yisaer))
- 优化 `TidbMonitor` 和 `TidbInitializer` controller 的日志 (#1493, [@aylei](https://github.com/aylei)
- 为 `Discovery Service` 和 `Discovery Deployment` 避免不必要的更新 (#1499, [@aylei](https://github.com/aylei))
- 移除某些没有意义的更新事件 (#1486, [@weekface](https://github.com/weekface))

12.1.22 TiDB Operator 1.1 Beta.1 Release Notes

发布日期：2020 年 1 月 8 日

TiDB Operator 版本：1.1.0-beta.1

12.1.22.1 需要操作的变更

- 所有 charts 支持配置 `timezone` (#1122, [@weekface](https://github.com/weekface))

  对于 `tidb-cluster` chart, 之前已经有 `timezone` 配置项 (默认值：`UTC`)。如果用户没有修改成不同值 (如：`Asia/Shangehai`)，所有 Pods 不会被重建。

  如果用户改成其它值 (如：`Asia/Shanghai`)，所有相关的 Pods (添加 `TZ` 环境变量) 会被重建 (滚动升级)。

  对于其它 charts，之前在对应的 `values.yaml` 里没有 `timezone` 配置项。这个 PR 添加了对 `timezone` 配置项的支持。不管用户使用旧的 `values.yaml` 还是新的 `values.yaml`，所有相关的 Pods (添加了 `TZ` 环境变量) 都不会被重建 (滚动升级)。

  相关的 Pods 有：`pump`，`drainer`，`discovery`，`monitor`，`scheduled backup`，`tidb-initializer`，`tikv-importer`。

  TiDB Operator 维护的全部镜像的 `timezone` 都是 `UTC`。如果你使用自己的镜像，请确保你的镜像的 `timezone` 是 `UTC`。

12.1.22.2 其他重要变更

- 支持使用 Backup & Restore (BR) 备份到 S3 (#1280, [@DanielZhangQD](https://github.com/Daniel
- 为 `TidbCluster` 添加基础默认设置及验证 (#1429, [@aylei](https://github.com/aylei))
- 支持使用增强型 StatefulSet 进行扩缩容 (#1361, [@cofyc](https://github.com/cofyc))
- 支持使用 `TidbInitializer` 自定义资源初始化 TiDB 集群 (#1403, [@DanielZhangQD](https://gith
- 优化 PD、TiKV、TiDB 的配置结构 (#1411, [@aylei](https://github.com/aylei))
- 设置 `tidbcluster` 拥有的资源的实例 label 键的默认名称为集群名 (#1419, [@aylei](https://github.com/aylei))
- `TidbCluster` 自定义资源支持管理 Pump 集群 (#1269, [@aylei](https://github.com/aylei))
- 修复 tikv-importer 默认配置中的错误 (#1415, [@aylei](https://github.com/aylei))
- 支持在资源配置中配置临时存储 (#1398, [@aylei](https://github.com/aylei))
- 添加不使用 Helm 运维 TiDB 集群的 e2e 测试用例 (#1396, [@aylei](https://github.com/aylei))
- 发布 Terraform Aliyun ACK 版本，指定默认版本为 `1.14.8-aliyun.1` (#1284, [@shonge](https://github.com/shonge))
- 优化 scheduler 的报错信息 (#1373, [@weekface](https://github.com/weekface))
- 把 `system:kube-scheduler` 集群 role 绑定到 `tidb-scheduler` 服务账号 (#1355, [@shonge](https://github.com/shonge))
- 添加新的 `TidbInitializer` 自定义资源类型 (#1391, [@aylei](https://github.com/aylei))
- 升级默认备份镜像为 `pingcap/tidb-cloud-backup:20191217`，优化 `-r` 选项 (#1360, [@aylei](https://github.com/aylei))
- 修复最新 EKS AMI 的 Docker ulimit 配置的错误 (#1349, [@aylei](https://github.com/aylei))
- 支持同步 Pump 状态到 TiDB 集群 (#1292, [@shonge](https://github.com/shonge))
- `tidb-controller-manager` 支持自动创建并调和 `tidb-discovery-service` (#1322, [@aylei](https://github.com/aylei))
- 扩大备份恢复的适用范围，提升安全性 (#1276, [@onlymellb](https://github.com/onlymellb))
- `TidbCluster` 自定义资源中添加 PD 和 TiKV 配置 (#1330, [@aylei](https://github.com/aylei))

- TidbCluster 自定义资源中添加 TiDB 配置 (#1291, [@aylei](https://github.com/aylei))
- 添加 TiKV 配置 schema (#1306, [@aylei](https://github.com/aylei))
- TiDB host:port 开启后再初始化 TiDB 集群，加快初始化速度 (#1296, [@cofyc](https://github.com/cofyc))
- 移除 DinD 相关脚本 (#1283, [@shonge](https://github.com/shonge))
- 支持从 AWS 和 GCP 的元数据获取验证证书 (#1248, [@gregwebs](https://github.com/gregwebs))
- 增加 tidb-controller-manager 的权限来操作 configmap (#1275, [@aylei](https://github.com/ayl
- 通过 tidb-controller-manager 管理 TiDB 服务 (#1242, [@aylei](https://github.com/aylei))
- 支持为组件设置 cluster-level 配置 (#1193, [@aylei](https://github.com/aylei))
- 从当前时间来获取时间字符串，代替从 Pod Name 获取 (#1229, [@weekface](https://github.com/weekface))
- 升级 TiDB 时，TiDB Operator 将不再注销 ddl owner，因为 TiDB 将在关闭时自动转移 ddl owner (#1239, [@aylei](https://github.com/aylei))
- 修复 Google terraform 模块的 use_ip_aliases 错误 (#1206, [@tennix](https://github.com/tennix))
- 升级 TiDB 默认版本为 v3.0.5 (#1179, [@shonge](https://github.com/shonge))
- 升级 Docker image 的基本系统到最新稳定版 (#1178, [@AstroProfundis](https://github.com/AstroP
- tkctl get TiKV 支持为每个 TiKV Pod 显示储存状态 (#916, [@Yisaer](https://github.com/Yisaer)
- 添加一个选项实现跨命名空间的监控 (#907, [@gregwebs](https://github.com/gregwebs))
- 在 tkctl get TiKV 中添加 STOREID 列，为每个 TiKV Pod 显示 store ID (#842, [@Yisaer](https://github.com/Yisaer))
- 用户可以在 chart 中通过 values.tidb.permitHost 指定被许可的主机 (#779, [@shonge](https://github.com/shonge))
- 为 kubelet 添加 zone 标签和资源预留配置 (#871, [@aylei](https://github.com/aylei))
- 修复了 apply 语法可能会导致 kubeconfig 被破坏的问题 (#861, [@cofyc](https://github.com/cofyc))
- 支持 TiKV 组件的灰度发布 (#869, [@onlymellb](https://github.com/onlymellb))
- 最新的 chart 兼容旧的 controller manager (#856, [@onlymellb](https://github.com/onlymellb))
- 添加对 TiDB 集群中 TLS 加密连接的基础支持 (#750, [@AstroProfundis](https://github.com/Astro
- 支持为 tidb-operator chart 配置 nodeSelector、亲和力和容忍度 (#855, [@shonge](https://github.com
- 支持为 TiDB 集群的所有容器配置资源请求和限制 (#853, [@aylei](https://github.com/aylei))
- 支持使用 Kind (Kubernetes IN Docker) 建立测试环境 (#791, [@xiaojingchen](https://github.com/xi
- 支持使用 tidb-lightning chart 恢复特定的数据源 (#827, [@tennix](https://github.com/tennix))
- 添加 tikvGCLifeTime 选项 (#835, [@weekface](https://github.com/weekface))
- 更新默认的备份镜像到 pingcap/tidb-cloud-backup:20190828 (#846, [@aylei](https://github.com
- 修复 Pump/Drainer 的数据目录避免潜在的数据丢失 (#826, [@aylei](https://github.com/aylei))
- 修复了 tkctl 使用 -oyaml 或 -ojson 标志不输出任何内容的问题，支持查看特定 Pod 或 PV 的详细信息，改进 tkctl get 命令的输出 (#822, [@onlymellb](https://github.com/onlymellb))
- 为 mydumper 添加推荐选项：-t 16 -F 64 --skip-tz-utc (#828, [@weekface](https://github.com/weekface))
- 在 deploy /gcp 中支持单可用区和多可用区集群 (#809, [@cofyc](https://github.com/cofyc))
- 修复当默认备份名被使用时备份失败的问题 (#836, [@DanielZhangQD](https://github.com/Daniel
- 增加对 TiDB Lightning 的支持 (#817, [@tennix](https://github.com/tennix))
- 支持从指定的定时备份目录还原 TiDB 集群 (#804, [@onlymellb](https://github.com/onlymellb))
- 修复了一个 tkctl 日志中的异常 (#797, [@onlymellb](https://github.com/onlymellb))
- 在 PD/TiKV/TiDB 规范中添加 hostNetwork 字段，以便可以在主机网络中运行

TiDB 组件 (#774, [@cofyc](https://github.com/cofyc))
- 当 mdadm 和 RAID 在 GKE 上可用时，使用 mdadm 和 RAID 而不是 LVM (#789, [@gregwebs](https://github.com/gregwebs))
- 支持通过增加 PVC 存储大小来动态扩展云存储 PV (#772, [@tennix](https://github.com/tennix))
- 支持为 PD/TiDB/TiKV 节点池配置节点镜像类型 (#776, [@cofyc](https://github.com/cofyc))
- 添加脚本以删除 GKE 的未使用磁盘 (#771, [@gregwebs](https://github.com/gregwebs))
- 对 Pump 和 Drainer 增加 `binlog.pump.config` and `binlog.drainer.config` 配置项 (#693, [@weekface](https://github.com/weekface))
- 当 Pump 变为 "offline" 状态时，阻止 Pump 进程退出 (#769, [@weekface](https://github.com/weekfa
- 引入新的 `tidb-drainer` helm chart 以管理多个 Drainer (#744, [@aylei](https://github.com/aylei))
- 添加 backup-manager 工具以支持备份、还原和清除备份数据 (#694, [@onlymellb](https://github.com/onlymellb))
- 在 Pump/Drainer 的配置中添加 `affinity` 选项 (#741, [@weekface](https://github.com/weekface))
- 修复 TiKV failover 后某些情况下的 TiKV 缩容失败 (#726, [@onlymellb](https://github.com/onlym
- 修复 UpdateService 的错误处理 (#718, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 将 e2e 运行时间从 60m 减少到 20m (#713, [@weekface](https://github.com/weekface))
- 添加 `AdvancedStatefulset` 功能以使用增强型 StatefulSet 代替 Kubernetes 内置的 StatefulSet (#1108, [@cofyc](https://github.com/cofyc))
- 支持为 TiDB 集群自动生成证书 (#782, [@AstroProfundis](https://github.com/AstroProfundis))
- 支持备份到 GCS (#1127, [@onlymellb](https://github.com/onlymellb))
- 支持为 TiDB 配置 `net.ipv4.tcp_keepalive_time` 和 `net.core.somaxconn`，以及为 TiKV 配置 `net.core.somaxconn` (#1107, [@DanielZhangQD](https://github.com/DanielZhangQD)
- 为聚合的 apiserver 添加基本的 e2e 测试 (#1109, [@aylei](https://github.com/aylei))
- 添加 `enablePVReclaim` 选项，在 tidb-operator 缩容 TiKV 或 PD 时回收 PV (#1037, [@onlymellb](https://github.com/onlymellb))
- 统一所有兼容 S3 的存储以支持备份和还原 (#1088, [@onlymellb](https://github.com/onlymellb))
- 将 `podSecuriyContext` 的默认值设置为 nil (#1079, [@aylei](https://github.com/aylei))
- 在 `tidb-operator` chart 中添加 `tidb-apiserver` (#1083, [@aylei](https://github.com/aylei))
- 添加新组件 TiDB aggregated apiserver (#1048, [@aylei](https://github.com/aylei))
- 修复了当发行版名称是 `un-wanted` 时 `tkctl version` 不起作用的问题 (#1065, [@aylei](https://github.com/aylei))
- 支持暂停备份计划 (#1047, [@onlymellb](https://github.com/onlymellb))
- 修复了在 Terraform 输出中 TiDB Loadbalancer 为空的问题 (#1045, [@DanielZhangQD](https://git
- 修复了 AWS terraform 脚本中 `create_tidb_cluster_release` 变量不起作用的问题 (#1062, [@aylei](https://github.com/aylei))
- 在稳定性测试中，默认情况下启用 `ConfigMapRollout` (#1036, [@aylei](https://github.com/aylei))
- 迁移到使用 `app/v1` 并且不再支持 1.9 版本之前的 Kubernetes (#1012, [@Yisaer](https://github.com
- 暂停 AWS TiKV 自动缩放组的 `ReplaceUnhealthy` 流程 (#1014, [@aylei](https://github.com/aylei))
- 将 `tidb-monitor-reloader` 镜像更改为 `pingcap/tidb-monitor-reloader:v1.0.1` (#898, [@qiffang](https://github.com/qiffang))
- 添加一些 sysctl 内核参数设置以进行调优 (#1016, [@tennix](https://github.com/tennix))
- 备份计划支持设置最长保留时间 (#979, [@onlymellb](https://github.com/onlymellb))
- 将默认的 TiDB 版本升级到 v3.0.4 (#837, [@shonge](https://github.com/shonge))
- 在阿里云上修复 TiDB Operator 的定制值文件 (#971, [@DanielZhangQD](https://github.com/Dani
- 在 TiKV 中添加 `maxFailoverCount` 限制 (#965, [@weekface](https://github.com/weekface))

- 支持在 AWS Terraform 脚本中为 tidb-operator 设置自定义配置 (#946, [@aylei](https://github.com/aylei))
- 在 TiKV 容量不是 GiB 的倍数时，将其转换为 MiB (#942, [@cofyc](https://github.com/cofyc))
- 修复 Drainer 的配置错误 (#939, [@weekface](https://github.com/weekface))
- 支持使用自定义的 `values.yaml` 部署 TiDB Operator 和 TiDB 集群 (#959, [@DanielZhangQD](https://github.com/DanielZhangQD))
- 支持为 PD、TiKV 和 TiDB Pod 指定 `SecurityContext`，并为 AWS 启用 `tcp keepalive` (#915, [@aylei](https://github.com/aylei))

12.2 v1.0

12.2.1 TiDB Operator 1.0.7 Release Notes

Release date: June 16, 2020

TiDB Operator version: 1.0.7

12.2.1.1 Notable Changes

- Fix alert rules lost after rolling upgrade (#2715)
- Upgrade local volume provisioner to 2.3.4 (#1778)
- Fix operator failover config invalid (#1877)
- Remove unnecessary duplicated docs (#2100)
- Update doc links and image in readme (#2106)
- Emit events when PD failover (#1466)
- Fix some broken urls (#1501)
- Remove some not very useful update events (#1486)

12.2.2 TiDB Operator 1.0.6 Release Notes

Release date: December 27, 2019

TiDB Operator version: 1.0.6

12.2.2.1 v1.0.6 What's New

Action required: Users should migrate the configs in `values.yaml` of previous chart releases to the new `values.yaml` of the new chart. Otherwise, the monitor pods might fail when you upgrade the monitor with the new chart.

For example, configs in the old `values.yaml` file:

```
monitor:
  ...
  initializer:
    image: pingcap/tidb-monitor-initializer:v3.0.5
```

```
    imagePullPolicy: IfNotPresent
  ...
```

After migration, configs in the new `values.yaml` file should be as follows:

```
monitor:
  ...
  initializer:
    image: pingcap/tidb-monitor-initializer:v3.0.5
    imagePullPolicy: Always
    config:
      K8S_PROMETHEUS_URL: http://prometheus-k8s.monitoring.svc:9090
  ...
```

12.2.2.1.1 Monitor

- Enable alert rule persistence (#898)
- Add node & pod info in TiDB Grafana (#885)

12.2.2.1.2 TiDB Scheduler

- Refine scheduler error messages (#1373)

12.2.2.1.3 Compatibility

- Fix the compatibility issue in Kubernetes v1.17 (#1241)
- Bind the `system:kube-scheduler` ClusterRole to the `tidb-scheduler` service account (#1355)

12.2.2.1.4 TiKV Importer

- Fix the default `tikv-importer` configuration (#1415)

12.2.2.1.5 E2E

- Ensure pods unaffected when upgrading (#955)

12.2.2.1.6 CI

- Move the release CI script from Jenkins into the tidb-operator repository (#1237)
- Adjust the release CI script for the `release-1.0` branch (#1320)

12.2.3 TiDB Operator 1.0.5 Release Notes

Release date: December 11, 2019

TiDB Operator version: 1.0.5

12.2.3.1 v1.0.5 What's New

There is no action required if you are upgrading from v1.0.4.

12.2.3.1.1 Scheduled Backup

- Fix the issue that backup failed when `clusterName` is too long (#1229)

12.2.3.1.2 TiDB Binlog

- It is recommended that TiDB and Pump be deployed on the same node through the `affinity` feature and Pump be dispersed on different nodes through the `anti-affinity` feature. At most only one Pump instance is allowed on each node. We added a guide to the chart. (#1251)

12.2.3.1.3 Compatibility

- Fix `tidb-scheduler` RBAC permission in Kubernetes v1.16 (#1282)
- Do not set `DNSPolicy` if `hostNetwork` is disabled to keep backward compatibility (#1287)

12.2.3.1.4 E2E

- Fix e2e nil point dereference (#1221)

12.2.4 TiDB Operator 1.0.4 Release Notes

Release date: November 23, 2019

TiDB Operator version: 1.0.4

12.2.4.1 v1.0.4 What's New

12.2.4.1.1 Action Required

There is no action required if you are upgrading from v1.0.3.

12.2.4.1.2 Highlights

#1202 introduced `HostNetwork` support, which offers better performance compared to the Pod network. Check out our benchmark report for details.

> **Note:**
>
> Due to this issue of Kubernetes, the Kubernetes cluster must be one of the following versions to enable `HostNetwork` of the TiDB cluster:
>
> - `v1.13.11` or later
> - `v1.14.7` or later
> - `v1.15.4` or later
> - any version since `v1.16.0`

#1175 added the `podSecurityContext` support for TiDB cluster Pods. We recommend setting the namespaced kernel parameters for TiDB cluster Pods according to our Environment Recommendation.

New Helm chart `tidb-lightning` brings TiDB Lightning support for TiDB in Kubernetes. Check out the document for detailed user guide.

Another new Helm chart `tidb-drainer` brings multiple drainers support for TiDB Binlog in Kubernetes. Check out the document for detailed user guide.

12.2.4.1.3 Improvements

- Support HostNetwork (#1202)
- Support configuring sysctls for Pods and enable net.* (#1175)
- Add tidb-lightning support (#1161)
- Add new helm chart tidb-drainer to support multiple drainers (#1160)

12.2.4.2 Detailed Bug Fixes and Changes

- Add e2e scripts and simplify the e2e Jenkins file (#1174)
- Fix the pump/drainer data directory to avoid data loss caused by bad configuration (#1183)
- Add init sql case to e2e (#1199)
- Keep the instance label of drainer same with the TiDB cluster in favor of monitoring (#1170)
- Set `podSecuriyContext` to nil by default in favor of backward compatibility (#1184)

12.2.4.3 Additional Notes for Users of v1.1.0.alpha branch

For historical reasons, `v1.1.0.alpha` is a hot-fix branch and got this name by mistake. All fixes in that branch are cherry-picked to `v1.0.4` and the `v1.1.0.alpha` branch will be discarded to keep things clear.

We strongly recommend you to upgrade to `v1.0.4` if you are using any version under `v1.1.0.alpha`.

`v1.0.4` introduces the following fixes comparing to `v1.1.0.alpha.3`:

- Support HostNetwork (#1202)
- Add the permit host option for tidb-initializer job (#779)
- Fix drainer misconfiguration in tidb-cluster chart (#945)
- Set the default `externalTrafficPolicy` to be Local for TiDB services (#960)
- Fix tidb-operator crash when users modify sts upgrade strategy improperly (#969)
- Add the `maxFailoverCount` limit to TiKV (#976)
- Fix values file customization for tidb-operator on Aliyun (#983)
- Do not limit failover count when maxFailoverCount = 0 (#978)
- Suspend the `ReplaceUnhealthy` process for TiKV auto-scaling-group on AWS (#1027)
- Fix the issue that the `create_tidb_cluster_release` variable does not work (#1066))
- Add `v1` to statefulset apiVersions (#1056)
- Add timezone support (#1126)

12.2.5 TiDB Operator 1.0.3 Release Notes

Release date: November 13, 2019

TiDB Operator version: 1.0.3

12.2.5.1 v1.0.3 What's New

12.2.5.1.1 Action Required

ACTION REQUIRED: This release upgrades default TiDB version to `v3.0.5` which fixed a serious bug in TiDB. So if you are using TiDB `v3.0.4` or prior versions, you **must** upgrade to `v3.0.5`.

ACTION REQUIRED: This release adds the `timezone` support for all charts.

For existing TiDB clusters. If the `timezone` in `tidb-cluster/values.yaml` has been customized to other timezones instead of the default `UTC`, then upgrading tidb-operator will trigger a rolling update for the related pods.

The related pods include `pump`, `drainer`, `discovery`, `monitor`, `scheduled backup`, `tidb-initializer`, and `tikv-importer`.

The time zone for all images maintained by `tidb-operator` should be `UTC`. If you use your own images, you need to make sure that the corresponding time zones are `UTC`.

12.2.5.1.2 Improvements

- Add the `timezone` support for all containers of the TiDB cluster
- Support configuring resource requests and limits for all containers of the TiDB cluster

12.2.5.2 Detailed Bug Fixes and Changes

- Upgrade default TiDB version to `v3.0.5` (#1132)
- Add the `timezone` support for all containers of the TiDB cluster (#1122)
- Support configuring resource requests and limits for all containers of the TiDB cluster (#853)

12.2.6 TiDB Operator 1.0.2 Release Notes

Release date: November 1, 2019

TiDB Operator version: 1.0.2

12.2.6.1 v1.0.2 What's New

12.2.6.1.1 Action Required

The AWS Terraform script uses auto-scaling-group for all components (PD/TiKV/TiDB/monitor). When an ec2 instance fails the health check, the instance will be replaced. This is helpful for those applications that are stateless or use EBS volumes to store data.

But a TiKV Pod uses instance store to store its data. When an instance is replaced, all the data on its store will be lost. TiKV has to resync all data to the newly added instance. Though TiDB is a distributed database and can work when a node fails, resyncing data can cost much if the dataset is large. Besides, the ec2 instance may be recovered to a healthy state by rebooting.

So we disabled the auto-scaling-group's replacing behavior in `v1.0.2`.

Auto-scaling-group scaling process can also be suspended according to its documentation if you are using `v1.0.1` or prior versions.

12.2.6.1.2 Improvements

- Suspend ReplaceUnhealthy process for AWS TiKV auto-scaling-group
- Add a new VM manager `qm` in stability test
- Add `tikv.maxFailoverCount` limit to TiKV
- Set the default `externalTrafficPolicy` to be `Local` for TiDB service in AWS/GCP/Aliyun
- Add provider and module versions for AWS

12.2.6.1.3 Bug Fixes

- Fix the issue that tkctl version does not work when the release name is un-wanted
- Migrate statefulsets apiVersion to `app/v1` which fixes compatibility with Kubernetes 1.16 and above versions
- Fix the issue that the `create_tidb_cluster_release` variable in AWS Terraform script does not work
- Fix compatibility issues by adding `v1beta1` to statefulset apiVersions
- Fix the issue that TiDB Loadbalancer is empty in Terraform output
- Fix a compatibility issue of TiKV `maxFailoverCount`
- Fix Terraform providers version constraint issues for GCP and Aliyun
- Fix values file customization for tidb-operator on Aliyun
- Fix tidb-operator crash when users modify statefulset upgrade strategy improperly
- Fix drainer misconfiguration

12.2.6.2 Detailed Bug Fixes and Changes

- Fix the issue that tkctl version does not work when the release name is un-wanted (#1065)
- Fix the issue that the `create_tidb_cluster_release` variable in AWS terraform script does not work (#1062)
- Fix compatibility issues for (#1012): add `v1beta1` to statefulset apiVersions (#1054)
- Enable ConfigMapRollout by default in stability test (#1036)
- Fix the issue that TiDB Loadbalancer is empty in Terraform output (#1045)
- Migrate statefulsets apiVersion to `app/v1` which fixes compatibility with Kubernetes 1.16 and above versions (#1012)
- Only expect TiDB cluster upgrade to be complete when rolling back wrong configuration in stability test (#1030)
- Suspend ReplaceUnhealthy process for AWS TiKV auto-scaling-group (#1014)
- Add a new VM manager `qm` in stability test (#896)
- Fix provider versions constraint issues for GCP and Aliyun (#959)
- Fix values file customization for tidb-operator on Aliyun (#971)
- Fix a compatibility issue of TiKV `tikv.maxFailoverCount` (#977)
- Add `tikv.maxFailoverCount` limit to TiKV (#965)
- Fix tidb-operator crash when users modify statefulset upgrade strategy improperly (#912)
- Set the default `externalTrafficPolicy` to be `Local` for TiDB service in AWS/GCP/Aliyun (#947)
- Add note about setting PV reclaim policy to retain (#911)
- Fix drainer misconfiguration (#939)
- Add provider and module versions for AWS (#926)

12.2.7 TiDB Operator 1.0.1 Release Notes

Release date: September 17, 2019

TiDB Operator version: 1.0.1

12.2.7.1 v1.0.1 What's New

12.2.7.1.1 Action Required

- ACTION REQUIRED: We fixed a serious bug (#878) that could cause all `PD` and `TiKV` pods to be accidentally deleted when `kube-apiserver` fails. This would cause TiDB service outage. So if you are using `v1.0.0` or prior versions, you **must** upgrade to `v1.0.1`.
- ACTION REQUIRED: The backup tool image pingcap/tidb-cloud-backup uses a forked version of `Mydumper`. The current version `pingcap/tidb-cloud-backup :20190610` contains a serious bug that could result in a missing column in the exported data. This is fixed in #29. And the default image used now contains this fixed version. So if you are using the old version image for backup, you **must** upgrade to use `pingcap/tidb-cloud-backup:201908028` and do a new full backup to avoid potential data inconsistency.

12.2.7.1.2 Improvements

- Modularize GCP Terraform
- Add a script to remove orphaned k8s disks
- Support `binlog.pump.config`, `binlog.drainer.config` configurations for Pump and Drainer
- Set the resource limit for the `tidb-backup` job
- Add `affinity` to Pump and Drainer configurations
- Upgrade local-volume-provisioner to `v2.3.2`
- Reduce e2e run time from `60m` to `20m`
- Prevent the Pump process from exiting with `0` if the Pump becomes `offline`
- Support expanding cloud storage PV dynamically by increasing PVC storage size
- Add the `tikvGCLifeTime` option to do backup
- Add important parameters to `tikv.config` and `tidb.config` in `values.yaml`
- Support restoring the TiDB cluster from a specified scheduled backup directory
- Enable cloud storage volume expansion & label local volume
- Document and improve HA algorithm
- Support specifying the permit host in the `values.tidb.permitHost` chart
- Add the zone label and reserved resources arguments to kubelet
- Update the default backup image to `pingcap/tidb-cloud-backup:20190828`

12.2.7.1.3 Bug Fixes

- Fix the TiKV scale-in failure in some cases after the TiKV failover

- Fix error handling for UpdateService
- Fix some orphan pods cleaner bugs
- Fix the bug of setting the `StatefulSet` partition
- Fix ad-hoc full backup failure due to incorrect `claimName`
- Fix the offline Pump: the Pump process will exit with `0` if going offline
- Fix an incorrect condition judgment

12.2.7.2 Detailed Bug Fixes and Changes

- Clean up `tidb.pingcap.com/pod-scheduling` annotation when the pod is scheduled (#790)
- Update tidb-cloud-backup image tag (#846)
- Add the TiDB permit host option (#779)
- Add the zone label and reserved resources for nodes (#871)
- Fix some orphan pods cleaner bugs (#878)
- Fix the bug of setting the `StatefulSet` partition (#830)
- Add the `tikvGCLifeTime` option (#835)
- Add recommendations options to Mydumper (#828)
- Fix ad-hoc full backup failure due to incorrect `claimName` (#836)
- Improve `tkctl get` command output (#822)
- Add important parameters to TiKV and TiDB configurations (#786)
- Fix the issue that `binlog.drainer.config` is not supported in v1.0.0 (#775)
- Support restoring the TiDB cluster from a specified scheduled backup directory (#804)
- Fix `extraLabels` description in `values.yaml` (#763)
- Fix tkctl log output exception (#797)
- Add a script to remove orphaned K8s disks (#745)
- Enable cloud storage volume expansion & label local volume (#772)
- Prevent the Pump process from exiting with `0` if the Pump becomes `offline` (#769)
- Modularize GCP Terraform (#717)
- Support `binlog.pump.config` configurations for Pump and Drainer (#693)
- Remove duplicate key values (#758)
- Fix some typos (#738)
- Extend the waiting time of the `CheckManualPauseTiDB` process (#752)
- Set the resource limit for the `tidb-backup` job (#729)
- Fix e2e test compatible with v1.0.0 (#757)
- Make incremental backup test work (#764)
- Add retry logic for `LabelNodes` function (#735)
- Fix the TiKV scale-in failure in some cases (#726)
- Add affinity to Pump and Drainer (#741)
- Refine cleanup logic (#719)
- Inject a failure by pod annotation (#716)
- Update README links to point to correct `pingcap.com/docs` URLs for English and Chinese (#732)
- Document and improve HA algorithm (#670)

- Fix an incorrect condition judgment (#718)
- Upgrade local-volume-provisioner to v2.3.2 (#696)
- Reduce e2e test run time (#713)
- Fix Terraform GKE scale-out issues (#711)
- Update wording and fix format for v1.0.0 (#709)
- Update documents (#705)

12.2.8 TiDB Operator 1.0 GA Release Notes

Release date: July 30, 2019

TiDB Operator version: 1.0.0

12.2.8.1 v1.0.0 What's New

12.2.8.1.1 Action Required

- ACTION REQUIRED: `tikv.storeLabels` was removed from `values.yaml`. You can directly set it with `location-labels` in `pd.config`.
- ACTION REQUIRED: the `--features` flag of tidb-scheduler has been updated to the `key={true,false}` format. You can enable the feature by appending `=true`.
- ACTION REQUIRED: you need to change the configurations in `values.yaml` of previous chart releases to the new `values.yaml` of the new chart. Otherwise, the configurations will be ignored when upgrading the TiDB cluster with the new chart.

The `pd` section in old `values.yaml`:

```
pd:
  logLevel: info
  maxStoreDownTime: 30m
  maxReplicas: 3
```

The `pd` section in new `values.yaml`:

```
pd:
  config: |
    [log]
    level = "info"
    [schedule]
    max-store-down-time = "30m"
    [replication]
    max-replicas = 3
```

The `tikv` section in old `values.yaml`:

```
tikv:
  logLevel: info
  syncLog: true
  readpoolStorageConcurrency: 4
  readpoolCoprocessorConcurrency: 8
  storageSchedulerWorkerPoolSize: 4
```

The `tikv` section in new `values.yaml`:

```
tikv:
  config: |
    log-level = "info"
    [server]
    status-addr = "0.0.0.0:20180"
    [raftstore]
    sync-log = true
    [readpool.storage]
    high-concurrency = 4
    normal-concurrency = 4
    low-concurrency = 4
    [readpool.coprocessor]
    high-concurrency = 8
    normal-concurrency = 8
    low-concurrency = 8
    [storage]
    scheduler-worker-pool-size = 4
```

The `tidb` section in old `values.yaml`:

```
tidb:
  logLevel: info
  preparedPlanCacheEnabled: false
  preparedPlanCacheCapacity: 100
  txnLocalLatchesEnabled: false
  txnLocalLatchesCapacity: "10240000"
  tokenLimit: "1000"
  memQuotaQuery: "34359738368"
  txnEntryCountLimit: "300000"
  txnTotalSizeLimit: "104857600"
  checkMb4ValueInUtf8: true
  treatOldVersionUtf8AsUtf8mb4: true
  lease: 45s
  maxProcs: 0
```

The `tidb` section in new `values.yaml`:

```
tidb:
  config: |
    token-limit = 1000
    mem-quota-query = 34359738368
    check-mb4-value-in-utf8 = true
    treat-old-version-utf8-as-utf8mb4 = true
    lease = "45s"
    [log]
    level = "info"
    [prepared-plan-cache]
    enabled = false
    capacity = 100
    [txn-local-latches]
    enabled = false
    capacity = 10240000
    [performance]
    txn-entry-count-limit = 300000
    txn-total-size-limit = 104857600
    max-procs = 0
```

The `monitor` section in old `values.yaml`:

```
monitor:
  create: true
  ...
```

The `monitor` section in new `values.yaml`:

```
monitor:
  create: true
   initializer:
     image: pingcap/tidb-monitor-initializer:v3.0.5
     imagePullPolicy: IfNotPresent
   reloader:
     create: true
     image: pingcap/tidb-monitor-reloader:v1.0.0
     imagePullPolicy: IfNotPresent
     service:
       type: NodePort
  ...
```

Please check cluster configuration for detailed configuration.

12.2.8.1.2 Stability Test Cases Added

- Stop all etcds and kubelets

12.2.8.1.3 Improvements

- Simplify GKE SSD setup
- Modularization for AWS Terraform scripts
- Turn on the automatic failover feature by default
- Enable configmap rollout by default
- Enable stable scheduling by default
- Support multiple TiDB clusters management in Alibaba Cloud
- Enable AWS NLB cross zone load balancing by default

12.2.8.1.4 Bug Fixes

- Fix sysbench installation on bastion machine of AWS deployment
- Fix TiKV metrics monitoring in default setup

12.2.8.2 Detailed Bug Fixes and Changes

- Allow upgrading TiDB monitor along with TiDB version (#666)
- Specify the TiKV status address to fix monitoring (#695)
- Fix sysbench installation on bastion machine for AWS deployment (#688)
- Update the `git add upstream` command to use `https` in contributing document (#690)
- Stability cases: stop kubelet and etcd (#665)
- Limit test cover packages (#687)
- Enable nlb cross zone load balancing by default (#686)
- Add TiKV raftstore parameters (#681)
- Support multiple TiDB clusters management for Alibaba Cloud (#658)
- Adjust the `EndEvictLeader` function (#680)
- Add more logs (#676)
- Update feature gates to support `key={true,false}` syntax (#677)
- Fix the typo meke to make (#679)
- Enable configmap rollout by default and quote configmap digest suffix (#678)
- Turn automatic failover on (#667)
- Sets node count for default pool equal to total desired node count (#673)
- Upgrade default TiDB version to v3.0.1 (#671)
- Remove storeLabels (#663)
- Change the way to configure TiDB/TiKV/PD in charts (#638)
- Modularize for AWS terraform scripts (#650)
- Change the `DeferClose` function (#653)
- Increase the default storage size for Pump from 10Gi to 20Gi in response to `stop-write-at-available-space` (#657)
- Simplify local SDD setup (#644)

12.2.9 TiDB Operator 1.0 RC.1 Release Notes

Release date: July 12, 2019

TiDB Operator version: 1.0.0-rc.1

12.2.9.1 v1.0.0-rc.1 What's New

12.2.9.1.1 Stability test cases added

- Stop kube-proxy
- Upgrade tidb-operator

12.2.9.1.2 Improvements

- Get the TS first and increase the TiKV GC life time to 3 hours before the full backup
- Add endpoints list and watch permission for controller-manager
- Scheduler image is updated to use "k8s.gcr.io/kube-scheduler" which is much smaller than "gcr.io/google-containers/hyperkube". You must pre-pull the new scheduler image into your airgap environment before upgrading.
- Full backup data can be uploaded to or downloaded from Amazon S3
- The terraform scripts support manage multiple TiDB clusters in one EKS cluster.
- Add `tikv.storeLables` setting
- On GKE one can use COS for TiKV nodes with small data for faster startup
- Support force upgrade when PD cluster is unavailable.

12.2.9.1.3 Bug Fixes

- Fix unbound variable in the backup script
- Give kube-scheduler permission to update/patch pod status
- Fix tidb user of scheduled backup script
- Fix scheduled backup to ceph object storage
- Fix several usability problems for AWS terraform deployment
- Fix scheduled backup bug: segmentation fault when backup user's password is empty

12.2.9.2 Detailed Bug Fixes and Changes

- Segmentation fault when backup user's password is empty (#649)
- Small fixes for terraform AWS (#646)
- TiKV upgrade bug fix (#626)
- Improve the readability of some code (#639)
- Support force upgrade when PD cluster is unavailable (#631)
- Add new terraform version requirement to AWS deployment (#636)

- GKE local ssd provisioner for COS (#612)
- Remove TiDB version from build (#627)
- Refactor so that using the PD API avoids unnecessary imports (#618)
- Add `storeLabels` setting (#527)
- Update google-kubernetes-tutorial.md (#622)
- Multiple clusters management in EKS (#616)
- Add Amazon S3 support to the backup/restore features (#606)
- Pass TiKV upgrade case (#619)
- Separate slow log with TiDB server log by default (#610)
- Fix the problem of unbound variable in backup script (#608)
- Fix notes of tidb-backup chart (#595)
- Give kube-scheduler ability to update/patch pod status. (#611)
- Use kube-scheduler image instead of hyperkube (#596)
- Fix pull request template grammar (#607)
- Local SSD provision: reduce network traffic (#601)
- Add operator upgrade case (#579)
- Fix a bug that TiKV status is always upgrade (#598)
- Build without debugger symbols (#592)
- Improve error messages (#591)
- Fix tidb user of scheduled backup script (#594)
- Fix dt case bug (#571)
- GKE terraform (#585)
- Fix scheduled backup to Ceph object storage (#576)
- Add stop kube-scheduler/kube-controller-manager test cases (#583)
- Add endpoints list and watch permission for controller-manager (#590)
- Refine fullbackup (#570)
- Make sure go modules files are always tidy and up to date (#588)
- Local SSD on GKE (#577)
- Stop kube-proxy case (#556)
- Fix resource unit (#573)
- Give local-volume-provisioner pod a QoS of Guaranteed (#569)
- Check PD endpoints status when it's unhealthy (#545)

12.2.10 TiDB Operator 1.0 Beta.3 Release Notes

Release date: June 6, 2019

TiDB Operator version: 1.0.0-beta.3

12.2.10.1 v1.0.0-beta.3 What's New

12.2.10.1.1 Action Required

- ACTION REQUIRED: `nodeSelectorRequired` was removed from values.yaml.

- ACTION REQUIRED: Comma-separated values support in `nodeSelector` has been dropped, please use new-added `affinity` field which has a more expressive syntax.

12.2.10.1.2 A lot of stability cases added

- ConfigMap rollout
- One PD replicas
- Stop TiDB Operator itself
- TiDB stable scheduling
- Disaster tolerance and data regions disaster tolerance
- Fix many bugs of stability test

12.2.10.1.3 New Features

- Introduce ConfigMap rollout management. With the feature gate open, configuration file changes will be automatically applied to the cluster via a rolling update. Currently, the `scheduler` and `replication` configurations of PD can not be changed via ConfigMap rollout. You can use `pd-ctl` to change these values instead, see #487 for details.
- Support stable scheduling for pods of TiDB members in tidb-scheduler.
- Support adding additional pod annotations for PD/TiKV/TiDB, e.g. fluent-bit.io/parser.
- Support the affinity feature of k8s which can define the rule of assigning pods to nodes
- Allow pausing during TiDB upgrade

12.2.10.1.4 Documentation Improvement

- GCP one-command deployment
- Refine user guides
- Improve GKE, AWS, Aliyun guide

12.2.10.1.5 Pass User Acceptance Tests

12.2.10.1.6 Other improvements

- Upgrade default TiDB version to v3.0.0-rc.1
- Fix a bug in reporting assigned nodes of TiDB members
- `tkctl get` can show cpu usage correctly now
- Adhoc backup now appends the start time to the PVC name by default.
- Add the privileged option for TiKV pod
- `tkctl upinfo` can show nodeIP podIP port now
- Get TS and use it before full backup using mydumper
- Fix capabilities issue for `tkctl debug` command

12.2.10.2 Detailed Bug Fixes and Changes

- Add capabilities and privilege mode for debug container (#537)
- Note helm versions in deployment docs (#553)
- Split public and private subnets when using existing vpc (#530)
- Release v1.0.0-beta.3 (#557)
- GKE terraform upgrade to 0.12 and fix bastion instance zone to be region agnostic (#554)
- Get TS and use it before full backup using mydumper (#534)
- Add port podip nodeip to tkctl upinfo (#538)
- Fix disaster tolerance of stability test (#543)
- Add privileged option for TiKV pod template (#550)
- Use staticcheck instead of megacheck (#548)
- Refine backup and restore documentation (#518)
- Fix stability tidb pause case (#542)
- Fix tkctl get cpu info rendering (#536)
- Fix Aliyun tf output rendering and refine documents (#511)
- Make webhook configurable (#529)
- Add pods disaster tolerance and data regions disaster tolerance test cases (#497)
- Remove helm hook annotation for initializer job (#526)
- Add stable scheduling e2e test case (#524)
- Upgrade TiDB version in related documentations (#532)
- Fix a bug in reporting assigned nodes of TiDB members (#531)
- Reduce wait time and fix stability test (#525)
- Fix documentation usability issues in GCP document (#519)
- PD replicas 1 and stop tidb-operator (#496)
- Pause-upgrade stability test (#521)
- Fix restore script bug (#510)
- Retry truncating sst files upon failure (#484)
- Upgrade default TiDB to v3.0.0-rc.1 (#520)
- Add `--namespace` when creating backup secret (#515)
- New stability test case for ConfigMap rollout (#499)
- Fix issues found in Queeny's test (#507)
- Pause rolling-upgrade process of TiDB statefulset (#470)
- GKE terraform and guide (#493)
- Support the affinity feature of Kubernetes which defines the rule of assigning pods to nodes (#475)
- Support adding additional pod annotations for PD/TiKV/TiDB (#500)
- Document PD configuration issue (#504)
- Refine Aliyun and AWS cloud TiDB configurations (#492)
- Update wording and add note (#502)
- Support stable scheduling for TiDB (#477)
- Fix `make lint` (#495)
- Support updating configuration on the fly (#479)
- Update AWS deploy docs after testing (#491)

- Add release-note to pull_request_template.md (#490)
- Design proposal of stable scheduling in TiDB (#466)
- Update DinD image to make it possible to configure HTTP proxies (#485)
- Fix a broken link (#489)
- Fix typo (#483)

12.2.11 TiDB Operator 1.0 Beta.2 Release Notes

Release date: May 10, 2019

TiDB Operator version: 1.0.0-beta.2

12.2.11.1 v1.0.0-beta.2 What's New

12.2.11.1.1 Stability has been greatly enhanced

- Refactored e2e test
- Added stability test, 7x24 running

12.2.11.1.2 Greatly improved ease of use

- One-command deployment for AWS, Aliyun
- Minikube deployment for testing
- Tkctl cli tool
- Refactor backup chart for ease use
- Refine initializer job
- Grafana monitor dashboard improved, support multi-version
- Improved user guide
- Contributing documentation

12.2.11.1.3 Bug fixes

- Fix PD start script, add join file when startup
- Fix TiKV failover take too long
- Fix PD ha when replcias is less than 3
- Fix a tidb-scheduler acquireLock bug and emit event when scheduled failed
- Fix scheduler ha bug with defer deleting pods
- Fix a bug when using shareinformer without deepcopy

12.2.11.1.4 Other improvements

- Remove pushgateway from TiKV pod
- Add GitHub templates for issue reporting and PR
- Automatically set the scheduler K8s version
- Switch to go module
- Support slow log of TiDB

12.2.11.2 Detailed Bug Fixes and Changes

- Don't initialize when there is no tidb.password (#282)
- Fix join script (#285)
- Document tool setup and e2e test detail in `CONTRIBUTING.md` (#288)
- Update setup.md (#281)
- Support slow log tailing sidcar for TiDB instance (#290)
- Flexible tidb initializer job with secret set outside of helm (#286)
- Ensure SLOW_LOG_FILE env variable is always set (#298)
- Fix setup document description (#300)
- Refactor backup (#301)
- Abandon vendor and refresh go.sum (#311)
- Set the SLOW_LOG_FILE in the startup script (#307)
- Automatically set the scheduler K8s version (#313)
- TiDB stability test main function (#306)
- Add fault-trigger server (#312)
- Add ad-hoc backup and restore function (#316)
- Add scale & upgrade case functions (#309)
- Add slack (#318)
- Log dump when test failed (#317)
- Add fault-trigger client (#326)
- Monitor checker (#320)
- Add blockWriter case for inserting data (#321)
- Add scheduled-backup test case (#322)
- Port ddl test as a workload (#328)
- Use fault-trigger at e2e tests and add some log (#330)
- Add binlog deploy and check process (#329)
- Fix e2e can not make (#331)
- Multi TiDB cluster testing (#334)
- Fix backup test bugs (#335)
- Delete `blockWrite.go` and use `blockwrite.go` instead (#333)
- Remove vendor (#344)
- Add more checks for scale & upgrade (#327)
- Support more fault injection (#345)
- Rewrite e2e (#346)
- Add failover test (#349)

- Fix HA when the number of replicas are less than 3 (#351)
- Add fault-trigger service file (#353)
- Fix dind doc (#352)
- Add additionalPrintColumns for TidbCluster CRD (#361)
- Refactor stability main function (#363)
- Enable admin privilege for prom (#360)
- Update `README.md` with new info (#365)
- Build CLI (#357)
- Add extraLabels variable in tidb-cluster chart (#373)
- Fix TiKV failover (#368)
- Separate and ensure setup before e2e-build (#375)
- Fix `codegen.sh` and lock related dependencies (#371)
- Add sst-file-corruption case (#382)
- Use release name as default clusterName (#354)
- Add util class to support adding annotations to Grafana (#378)
- Use Grafana provisioning to replace dashboard installer (#388)
- Ensure test env is ready before cases running (#386)
- Remove monitor config job check (#390)
- Update local-pv documentation (#383)
- Update Jenkins links in `README.md` (#395)
- Fix e2e workflow in `CONTRIBUTING.md` (#392)
- Support running stability test out of cluster (#397)
- Update TiDB secret docs and charts (#398)
- Enable blockWriter write pressure in stability test (#399)
- Support debug and ctop commands in CLI (#387)
- Marketplace update (#380)
- Update editable value from true to false (#394)
- Add fault inject for kube proxy (#384)
- Use `ioutil.TempDir()` create charts and operator repo's directories (#405)
- Improve workflow in docs/google-kubernetes-tutorial.md (#400)
- Support plugin start argument for TiDB instance (#412)
- Replace govet with official vet tool (#416)
- Allocate 24 PVs by default (after 2 clusters are scaled to (#407)
- Refine stability (#422)
- Record event as grafana annotation in stability test (#414)
- Add GitHub templates for issue reporting and PR (#420)
- Add TiDBUpgrading func (#423)
- Fix operator chart issue (#419)
- Fix stability issues (#433)
- Change cert generate method and add pd and kv prestop webhook (#406)
- A tidb-scheduler bug fix and emit event when scheduled failed (#427)
- Shell completion for tkctl (#431)
- Delete an duplicate import (#434)
- Add etcd and kube-apiserver faults (#367)
- Fix TiDB Slack link (#444)

- Fix scheduler ha bug (#443)
- Add terraform script to auto deploy TiDB cluster on AWS (#401)
- Add instructions to access Grafana in GKE tutorial (#448)
- Fix label selector (#437)
- No need to set ClusterIP when syncing headless service (#432)
- Document how to deploy TiDB cluster with tidb-operator in minikube (#451)
- Add slack notify (#439)
- Fix local dind env (#440)
- Add terraform scripts to support alibaba cloud ACK deployment (#436)
- Fix backup data compare logic (#454)
- Async emit annotations (#438)
- Use TiDB v2.1.8 by default & remove pushgateway (#435)
- Fix a bug that uses shareinformer without copy (#462)
- Add version command for tkctl (#456)
- Add tkctl user manual (#452)
- Fix binlog problem on large scale (#460)
- Copy kubernetes.io/hostname label to PVs (#464)
- AWS EKS tutorial change to new terraform script (#463)
- Update documentation of minikube installation (#471)
- Update documentation of DinD installation (#458)
- Add instructions to access Grafana (#476)
- Support-multi-version-dashboard (#473)
- Update Aliyun deploy docs after testing (#474)
- GKE local SSD size warning (#467)
- Update roadmap (#376)

12.2.12 TiDB Operator 1.0 Beta.1 P2 Release Notes

Release date: February 21, 2019

TiDB Operator version: 1.0.0-beta.1-p2

12.2.12.1 Notable Changes

- New algorithm for scheduler HA predicate (#260)
- Add TiDB discovery service (#262)
- Serial scheduling (#266)
- Change tolerations type to an array (#271)
- Start directly when where is join file (#275)
- Add code coverage icon (#272)
- In `values.yml`, omit just the empty leaves (#273)
- Charts: backup to ceph object storage (#280)
- Add `ClusterIDLabelKey` label to TidbCluster (#279)

12.2.13 TiDB Operator 1.0 Beta.1 P1 Release Notes

Release date: January 7, 2019

TiDB Operator version: 1.0.0-beta.1-p1

12.2.13.1 Bug Fixes

- Fix scheduler policy issue, works on kubernetes v1.10, v1.11 and v1.12 now (#256)

12.2.13.2 Docs

- Proposal: add multiple statefulsets support to TiDB Operator (#240)
- Update roadmap (#258)

12.2.14 TiDB Operator 1.0 Beta.1 Release Notes

Release date: December 27, 2018

TiDB Operator version: 1.0.0-beta.1

12.2.14.1 Bug Fixes

- Fix pd_control bug: avoid relying on PD error response text (#197)
- Add orphan pod cleaner (#201)
- Fix scheduler configuration for Kubernetes 1.12 (#200)
- Fix Grafana configuration (#206)
- Fix pd failover bug: scale out directly when failover occurs (#217)
- Refactor PD failover (#211)
- Refactor tidb_cluster_control logic (#215)
- Fix upgrade logic: avoid updating pd/tikv/tidb simultaneously (#234)
- Fix PD control logic: get member/store before delete member/store and fix member id parse error (#245)
- Fix documents errors (#213)
- Fix backup and restore script bug (#251 #254 #255)
- Fix GKE multiple availability zones deployment PD disk scheduling bug (#248)

12.2.14.2 Minor Improvements

- Add Kubernetes 1.12 local DinD scripts (#195)
- Bump default TiDB to v2.1.0 (#212)
- Release tidb-operator/tidb-cluster charts (#216)
- Add connection timeout for TiDB password setter job (#219)

- Separate ad-hoc backup and restore to another chart (#227)
- Add compiler version info to tidb-operator binary (#237)
- Allow specifying TiDB service LoadBalancer IP (#246)
- Expose TiKV cpu/memory related configuration to values.yaml (#252)

12.2.15 TiDB Operator 1.0 Beta.0 Release Notes

Release date: November 26, 2018

TiDB Operator version: 1.0.0-beta.0

12.2.15.1 Notable Changes

- Introduce basic chaos testing
- Improve unit test coverage (#179 #181 #182 #184 #190 #192 #194)
- Add default value for log-level of PD/TiKV/TiDB (#185)
- Fix PD connection timeout issue for DinD environment (#186)
- Fix monitor configuration (#193)
- Fix document Helm client version requirement (#175)
- Keep scheduler name consistent in chart (#188)
- Remove unnecessary warning message when volumeName is empty (#177)
- Migrate to Go 1.11 module (#178)
- Add user guide (#187)

12.3 v0

12.3.1 TiDB Operator 0.4 Release Notes

Release date: November 9, 2018

TiDB Operator version: 0.4.0

12.3.1.1 Notable Changes

- Extend Kubernetes built-in scheduler for TiDB data awareness pod scheduling (#145)
- Restore backup data from GCS bucket (#160)
- Set password for TiDB when a TiDB cluster is first deployed (#171)

12.3.1.2 Minor Changes and Bug Fixes

- Update roadmap for the following two months (#166)
- Add more unit tests (#169)
- E2E test with multiple clusters (#162)

- E2E test for meta info synchronization (#164)
- Add TiDB failover limit (#163)
- Synchronize PV reclaim policy early to persist data (#169)
- Use helm release name as instance label (#168) (breaking change)
- Fix local PV setup script (#172)

12.3.2 TiDB Operator 0.3.1 Release Notes

Release date: October 31, 2018

TiDB Operator version: 0.3.1

12.3.2.1 Minor Changes

- Paramertize the serviceAccount (#116 #111)
- Bump TiDB to v2.0.7 & allow user specified config files (#121)
- Remove binding mode for GKE pd-ssd storageclass (#130)
- Modified placement of tidb_version (#125)
- Update google-kubernetes-tutorial.md (#105)
- Remove redundant creation statement of namespace tidb-operator-e2e (#132)
- Update the label name of app in local dind documentation (#136)
- Remove noisy events (#131)
- Marketplace (#123 #135)
- Change monitor/backup/binlog pvc labels (#143)
- TiDB readiness probes (#147)
- Add doc on how to provision kubernetes on AWS (#71)
- Add imagePullPolicy support (#152)
- Separation startup scripts and application config from yaml files (#149)
- Update marketplace for our open source offering (#151)
- Add validation to crd (#153)
- Marketplace: use the Release.Name (#157)

12.3.2.2 Bug Fixes

- Fix parallel upgrade bug (#118)
- Fix wrong parameter AGRS to ARGS (#114)
- Can't recover after a upgrade failed (#120)
- Scale in when store id match (#124)
- PD can't scale out if not all members are ready (#142)
- podLister and pvcLister usages are wrong (#158)

12.3.3 TiDB Operator 0.3.0 Release Notes

Release date: October 12, 2018

TiDB Operator version: 0.3.0

12.3.3.1 Notable Changes

- Add full backup support
- Add TiDB Binlog support
- Add graceful upgrade feature
- Allow monitor data to be persistent

12.3.4 TiDB Operator 0.2.1 Release Notes

Release date: September 20, 2018

TiDB Operator version: 0.2.1

12.3.4.1 Bug Fixes

- Fix retry on conflict logic (#87)
- Fix TiDB timezone configuration by setting TZ environment variable (#96)
- Fix failover by keeping spec replicas unchanged (#95)
- Fix repeated updating pod and pd/tidb StatefulSet (#101)

12.3.5 TiDB Operator 0.2.0 Release Notes

Release date: September 11, 2018

TiDB Operator version: 0.2.0

12.3.5.1 Notable Changes

- Support auto-failover experimentally
- Unify Tiller managed resources and TiDB Operator managed resources labels (breaking change)
- Manage TiDB service via Tiller instead of TiDB Operator, allow more parameters to be customized (required for public cloud load balancer)
- Add toleration for TiDB cluster components (useful for dedicated deployment)
- Add script to easy setup DinD environment
- Lint and format code in CI
- Refactor upgrade functions as interface

12.3.6 TiDB Operator 0.1.0 Release Notes

Release date: August 22, 2018

TiDB Operator version: 0.1.0

12.3.6.1 Notable Changes

- Bootstrap multiple TiDB clusters
- Monitor deployment support
- Helm charts support
- Basic Network PV/Local PV support
- Safely scale the TiDB cluster
- Upgrade the TiDB cluster in order
- Stop the TiDB process without terminating Pod
- Synchronize cluster meta info to POD/PV/PVC labels
- Basic unit tests & E2E tests
- Tutorials for GKE, local DinD
