TiDB in Kubernetes 用户文档

PingCAP Inc.

20230320

Table of Contents

1 TiDB in Kubernetes 文档 **9**

2 TiDB Operator 简介 **9**

2.1 TiDB Operator 整体架构 · · · · · 10

2.2 使用 TiDB Operator 管理 TiDB 集群 · · · · · 11

3 Benchmark **12**

3.1 TiDB in Kubernetes Sysbench 性能测试 · · · · · 12

3.1.1 目的 · · · · · 13

3.1.2 环境 · · · · · 13

3.1.3 测试报告 · · · · · 17

3.1.4 结语 · · · · · 33

4 快速上手 **34**

4.1 在 GCP 上通过 Kubernetes 部署 TiDB 集群 · · · · · 34

4.1.1 选择一个项目 · · · · · 34

4.1.2 启用 API · · · · · 34

4.1.3 配置 gcloud · · · · · 34

4.1.4 启动 3 个节点的 Kubernetes 集群 · · · · · 35

4.1.5 安装 Helm · · · · · 35

4.1.6 添加 Helm 仓库 · · · · · 36

4.1.7 部署 TiDB 集群 · · · · · 36

4.1.8 部署你的第一个 TiDB 集群 36
4.1.9 访问 TiDB 集群 37
4.1.10 扩容 TiDB 集群 38
4.1.11 访问 Grafana 面板 38
4.1.12 销毁 TiDB 集群 38
4.1.13 删除 Kubernetes 集群 38
4.1.14 更多信息 39
4.2 在 Minikube 集群上部署 TiDB 集群 39
4.2.1 安装 Minikube 并启动 Kubernetes 集群 39
4.2.2 安装 kubectl 并访问集群 40
4.2.3 安装 TiDB Operator 并运行 TiDB 集群 40
4.2.4 测试 TiDB 集群 42
4.2.5 监控 TiDB 集群 42
4.2.6 FAQ 43

5 部署 43

5.1 Kubernetes 上的 TiDB 集群环境需求 43
5.1.1 软件版本要求 43
5.1.2 内核参数设置 44
5.1.3 硬件和部署要求 45
5.1.4 Kubernetes 系统资源要求 46
5.1.5 TiDB 集群资源需求 46
5.1.6 TiDB 集群规划示例 47
5.2 在 Kubernetes 上部署 TiDB Operator 47
5.2.1 准备环境 48
5.2.2 部署 Kubernetes 集群 48
5.2.3 安装 Helm 49
5.2.4 配置本地持久化卷 49
5.2.5 安装 TiDB Operator 49
5.2.6 自定义 TiDB Operator 50

5.3 在标准 Kubernetes 上部署 TiDB 集群 50
5.3.1 前置条件 51
5.3.2 配置 TiDB 集群 51
5.3.3 部署 TiDB 集群 52
5.4 在 AWS EKS 上部署 TiDB 集群 53
5.4.1 环境配置准备 53
5.4.2 部署集群 54
5.4.3 访问数据库 55
5.4.4 Grafana 监控 56
5.4.5 升级 TiDB 集群 56
5.4.6 扩容 TiDB 集群 56
5.4.7 自定义 57
5.4.8 管理多个 TiDB 集群 59
5.4.9 仅管理基础设施 60
5.4.10 销毁集群 61
5.4.11 管理多个 Kubernetes 集群 61
5.5 在 GCP GKE 上部署 TiDB 集群 64
5.5.1 环境准备 65
5.5.2 配置 65
5.5.3 部署 TiDB 集群 66
5.5.4 访问 TiDB 数据库 68
5.5.5 与 GKE 集群交互 68
5.5.6 升级 TiDB 集群 69
5.5.7 管理多个 TiDB 集群 70
5.5.8 扩容 71
5.5.9 自定义 72
5.5.10 销毁 TiDB 集群 75
5.5.11 管理多个 Kubernetes 集群 76
5.6 在阿里云上部署 TiDB 集群 80
5.6.1 环境需求 80
5.6.2 概览 81
5.6.3 安装部署 81

5.6.4 连接数据库 82
5.6.5 监控 83
5.6.6 升级 TiDB 集群 83
5.6.7 TiDB 集群水平伸缩 83
5.6.8 销毁集群 83
5.6.9 配置 84
5.6.10 管理多个 TiDB 集群 84
5.6.11 管理多个 Kubernetes 集群 86
5.6.12 使用限制 87
5.7 访问 Kubernetes 上的 TiDB 集群 87
5.7.1 NodePort 87
5.7.2 LoadBalancer 88
5.8 TiDB Binlog 运维 88
5.8.1 运维准备 89
5.8.2 启用 TiDB 集群的 TiDB Binlog 89
5.8.3 部署多个 drainer 92

6 配置 93

6.1 Kubernetes 上的集群初始化配置 93
6.1.1 设置初始化账号和密码 93
6.1.2 批量执行初始化 SQL 语句 94
6.2 Kubernetes 上的 TiDB 集群配置 95
6.2.1 配置参数 95
6.2.2 资源配置说明 145
6.2.3 容灾配置说明 146
6.3 Kubernetes 上的 TiDB 集群备份配置 148
6.3.1 `mode` 148
6.3.2 `clusterName` 148
6.3.3 `name` 148
6.3.4 `secretName` 148
6.3.5 `storage.className` 149
6.3.6 `storage.size` 149

6.3.7 backupOptions 149
6.3.8 restoreOptions 149
6.3.9 gcp.bucket 149
6.3.10 gcp.secretName 149
6.3.11 ceph.endpoint 150
6.3.12 ceph.bucket 150
6.3.13 ceph.secretName 150
6.4 Kubernetes 上的持久化存储类型配置 151
6.4.1 TiDB 集群推荐存储类型 151
6.4.2 网络 PV 配置 152
6.4.3 本地 PV 配置 152
6.4.4 数据安全 156
6.5 Kubernetes 上的 TiDB Binlog Drainer 配置 157
6.5.1 配置参数 157

7 Kubernetes 上的 TiDB 集群监控 162
7.1 TiDB 集群的监控 162
7.1.1 查看监控面板 162
7.1.2 访问监控数据 162
7.2 Kubernetes 的监控 163
7.2.1 宿主机监控 163
7.2.2 Kubernetes 组件监控 164
7.3 报警配置 164
7.3.1 TiDB 集群报警 164
7.3.2 Kubernetes 报警 164

8 运维 164
8.1 销毁 Kubernetes 上的 TiDB 集群 164
8.2 重启 Kubernetes 上的 TiDB 集群 165
8.2.1 强制重启某个 Pod 165
8.2.2 强制重启某个组件的所有 Pod 166
8.2.3 强制重启 TiDB 集群的所有 Pod 166

8.3 维护 TiDB 集群所在的 Kubernetes 节点 166
8.3.1 维护 PD 和 TiDB 实例所在节点 167
8.3.2 维护 TiKV 实例所在节点 168
8.4 使用 PD Recover 恢复 PD 集群 170
8.4.1 下载 PD Recover 170
8.4.2 使用 PD Recover 恢复 PD 集群 170
8.5 备份与恢复 173
8.5.1 基于 Helm Charts 实现的 TiDB 集群备份与恢复 173
8.5.2 恢复 Kubernetes 上的 TiDB 集群数据 176
8.6 日志收集 179
8.6.1 TiDB 与 Kubernetes 组件运行日志 179
8.6.2 TiDB 慢查询日志 180
8.6.3 系统日志 181
8.7 Kubernetes 上的 TiDB 集群故障自动转移 181
8.7.1 实现原理 182

9 Kubernetes 上的 TiDB 集群扩缩容 183
9.1 水平扩缩容 183
9.1.1 水平扩缩容操作 183
9.2 垂直扩缩容 184
9.2.1 垂直扩缩容操作 184

10 升级 185
10.1 滚动升级 Kubernetes 上的 TiDB 集群 185
10.1.1 升级 TiDB 版本 185
10.1.2 更新 TiDB 集群配置 186
10.1.3 强制升级 TiDB 集群 187
10.2 升级 TiDB Operator 187
10.2.1 升级步骤 187
10.2.2 升级 Kubernetes 188

11 工具 188

11.1 tkctl 使用指南 188
11.1.1 安装 188
11.1.2 命令自动补全 189
11.1.3 Kubernetes 配置 189
11.1.4 所有命令 189
11.2 Kubernetes 上的 TiDB 工具指南 195
11.2.1 在 Kubernetes 上使用 PD Control 195
11.2.2 在 Kubernetes 上使用 TiKV Control 196
11.2.3 在 Kubernetes 上使用 TiDB Control 197
11.2.4 使用 Helm 197
11.2.5 使用 Terraform 199

12 组件 199
12.1 TiDB Scheduler 扩展调度器 199
12.1.1 TiDB 集群调度需求 199
12.1.2 工作原理 201

13 Kubernetes 上的 TiDB 集群故障诊断 202
13.1 诊断模式 202
13.2 集群意外删除后恢复 202
13.3 Pod 未正常创建 203
13.4 Pod 之间网络不通 203
13.5 Pod 处于 Pending 状态 204
13.6 Pod 处于 CrashLoopBackOff 状态 205
13.7 无法访问 TiDB 服务 205
13.8 TiKV Store 异常进入 Tombstone 状态 207
13.9 TiDB 长连接被异常中断 208

14 Kubernetes 上的 TiDB 集群常见问题 208
14.1 如何修改时区设置？ 209
14.2 TiDB 相关组件可以配置 HPA 或 VPA 么？ 209
14.3 使用 TiDB Operator 编排 TiDB 集群时，有什么场景需要人工介入操作吗？ 209
14.4 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？ 209

14.5 TiDB Operator 支持 TiSpark 吗？ 210
14.6 如何查看 TiDB 集群配置？ 210
14.7 部署 TiDB 集群时调度失败是什么原因？ 210

15 版本发布历史 211

15.1 v1.0 211
15.1.1 TiDB Operator 1.0.7 Release Notes 211
15.1.2 TiDB Operator 1.0.6 Release Notes 211
15.1.3 TiDB Operator 1.0.5 Release Notes 212
15.1.4 TiDB Operator 1.0.4 Release Notes 213
15.1.5 TiDB Operator 1.0.3 Release Notes 215
15.1.6 TiDB Operator 1.0.2 Release Notes 216
15.1.7 TiDB Operator 1.0.1 Release Notes 217
15.1.8 TiDB Operator 1.0 GA Release Notes 220
15.1.9 TiDB Operator 1.0 RC.1 Release Notes 223
15.1.10 TiDB Operator 1.0 Beta.3 Release Notes 225
15.1.11 TiDB Operator 1.0 Beta.2 Release Notes 228
15.1.12 TiDB Operator 1.0 Beta.1 P2 Release Notes 231
15.1.13 TiDB Operator 1.0 Beta.1 P1 Release Notes 232
15.1.14 TiDB Operator 1.0 Beta.1 Release Notes 232
15.1.15 TiDB Operator 1.0 Beta.0 Release Notes 233
15.2 v0 233
15.2.1 TiDB Operator 0.4 Release Notes 233
15.2.2 TiDB Operator 0.3.1 Release Notes 234
15.2.3 TiDB Operator 0.3.0 Release Notes 234
15.2.4 TiDB Operator 0.2.1 Release Notes 235
15.2.5 TiDB Operator 0.2.0 Release Notes 235
15.2.6 TiDB Operator 0.1.0 Release Notes 235

1 TiDB in Kubernetes 文档

2 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。

> **注意：**
>
> 每个 Kubernetes 集群中只能部署一个 TiDB Operator。

TiDB Operator 与适用的 TiDB 版本的对应关系如下：

TiDB 版本	适用的 TiDB Operator 版本
dev	dev
TiDB >= 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 TiDB Operator 整体架构

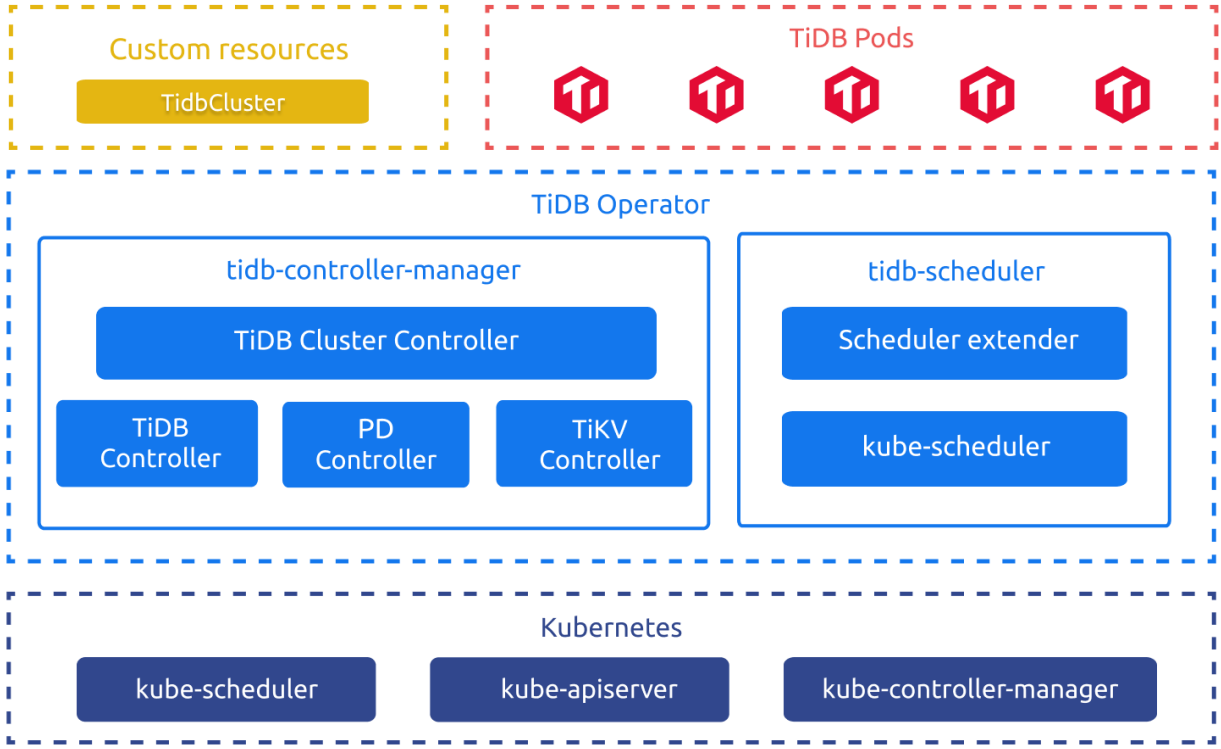


Figure 1: TiDB Operator Overview

其中，`TidbCluster` 是由 CRD（`CustomResourceDefinition`）定义的自定义资源，用于描述用户期望的 TiDB 集群状态。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；
- `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。

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。

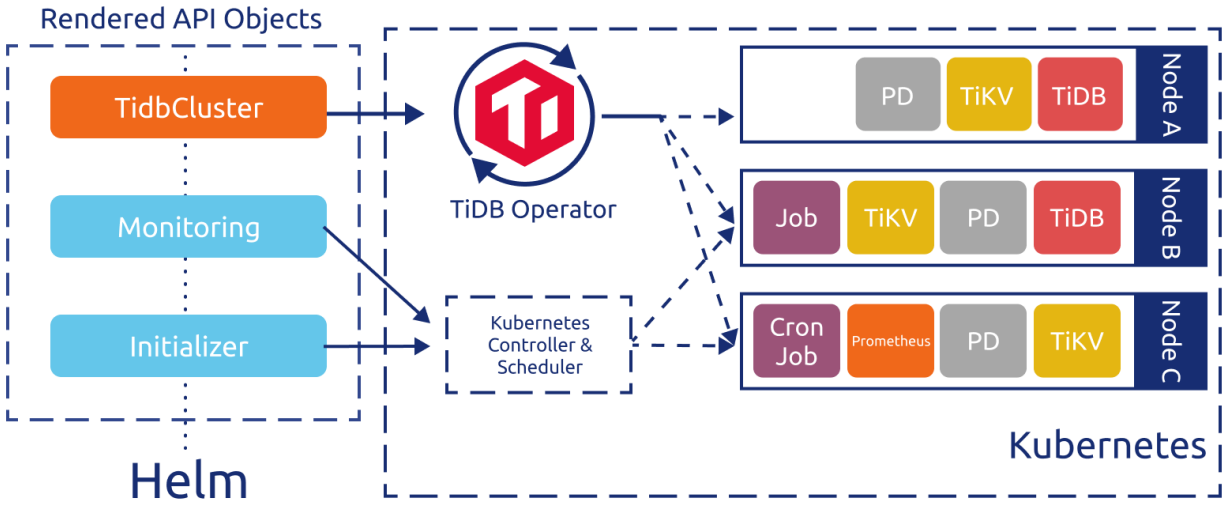


Figure 2: TiDB Operator Control Flow

上图是 TiDB Operator 的控制流程解析。由于 TiDB 集群还需要监控、初始化、定时备份、Binlog 等组件，TiDB Operator 中使用 Helm Chart 封装了 TiDB 集群定义。整体的控制流程如下：

1. 用户通过 Helm 创建 `TidbCluster` 对象和相应的一系列 Kubernetes 原生对象，比如执行定时备份的 `CronJob`；
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 的 `StatefulSet` 和 `Service` 对象；
3. Kubernetes 的原生控制器根据 `StatefulSet`、`Deployment`、`CronJob` 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。

2.2 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：
  - Minikube：使用 TiDB Operator 在本地 Minikube 环境部署 TiDB 集群；
  - GKE：使用 TiDB Operator 在 GKE 上部署 TiDB 集群。
- 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 集群备份恢复
- 配置 TiDB 集群故障自动转移
- 监控 TiDB 集群
- TiDB 集群日志收集
- 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。

3 Benchmark

3.1 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。

3.1.1 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

3.1.2 环境

3.1.2.1 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
```

3.1.2.2 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;
```

3.1.2.3 硬件配置

3.1.2.3.1 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。

3.1.2.3.2 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]
```

3.1.2.3.3 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。

3.1.2.3.4 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。

3.1.2.4 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。

3.1.2.5 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。

3.1.2.5.1 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)。

3.1.2.5.2 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
```

3.1.2.5.3 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。

3.1.3 测试报告

3.1.3.1 单可用区测试

3.1.3.1.1 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：

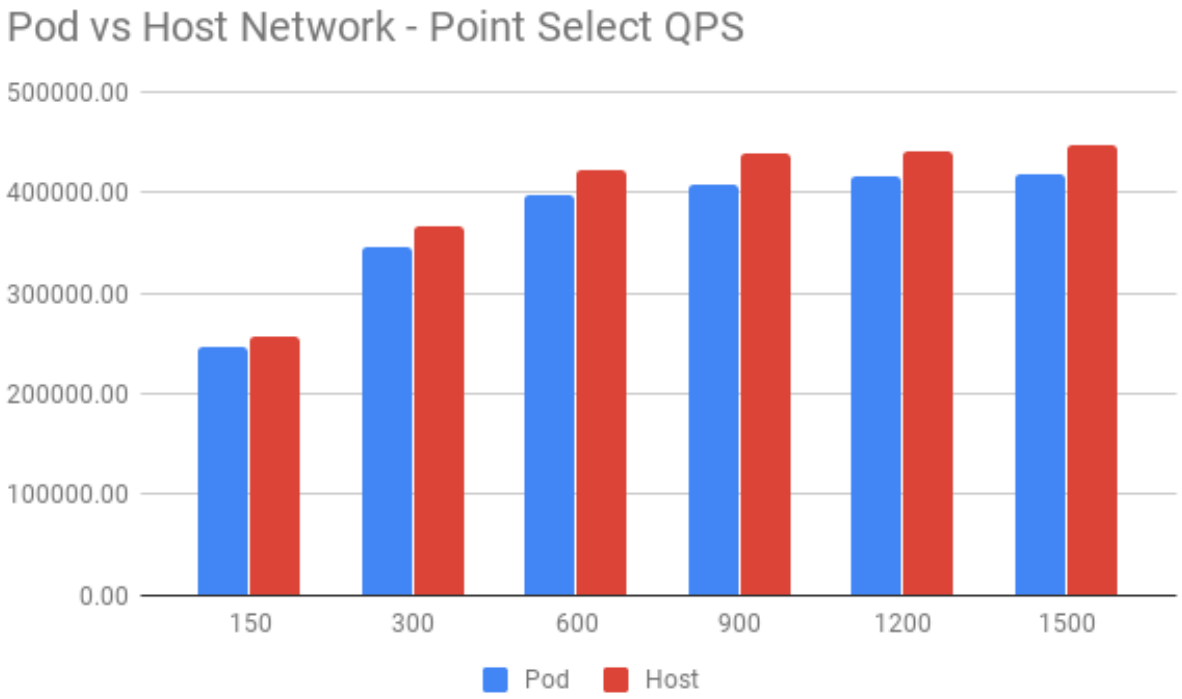


Figure 3: Pod vs Host Network

Latency 对比：

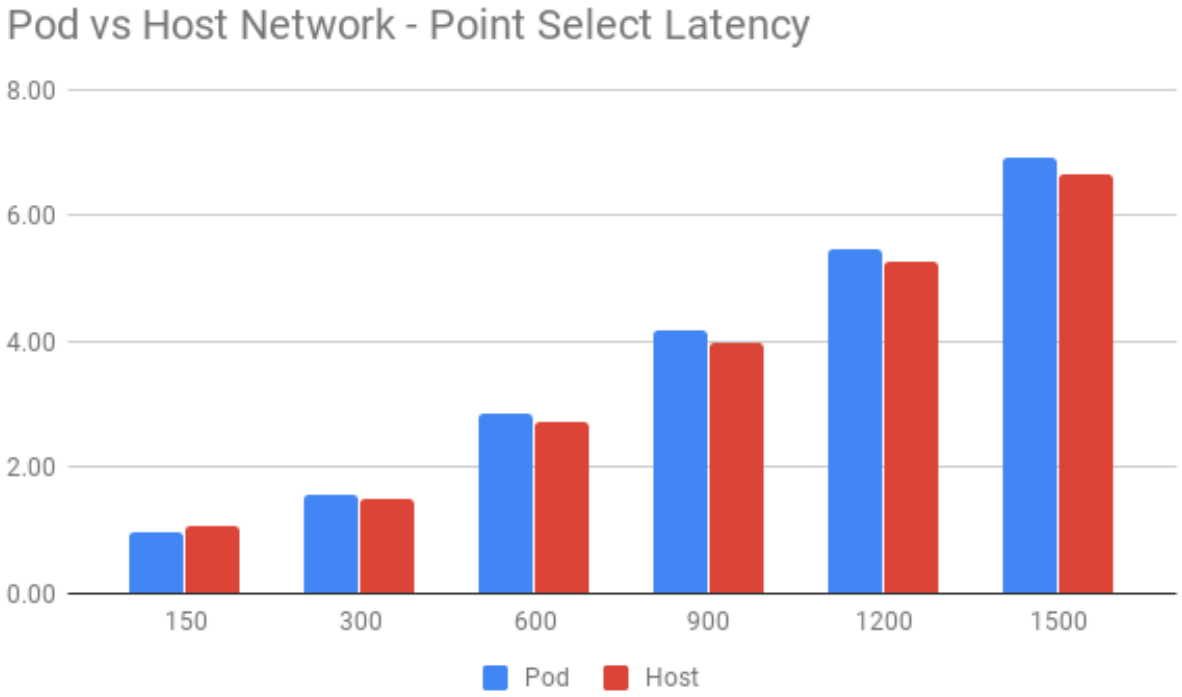


Figure 4: Pod vs Host Network

从图中可以看到 Host 网络下整体表现略好于 Pod 网络。

3.1.3.1.2 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：**

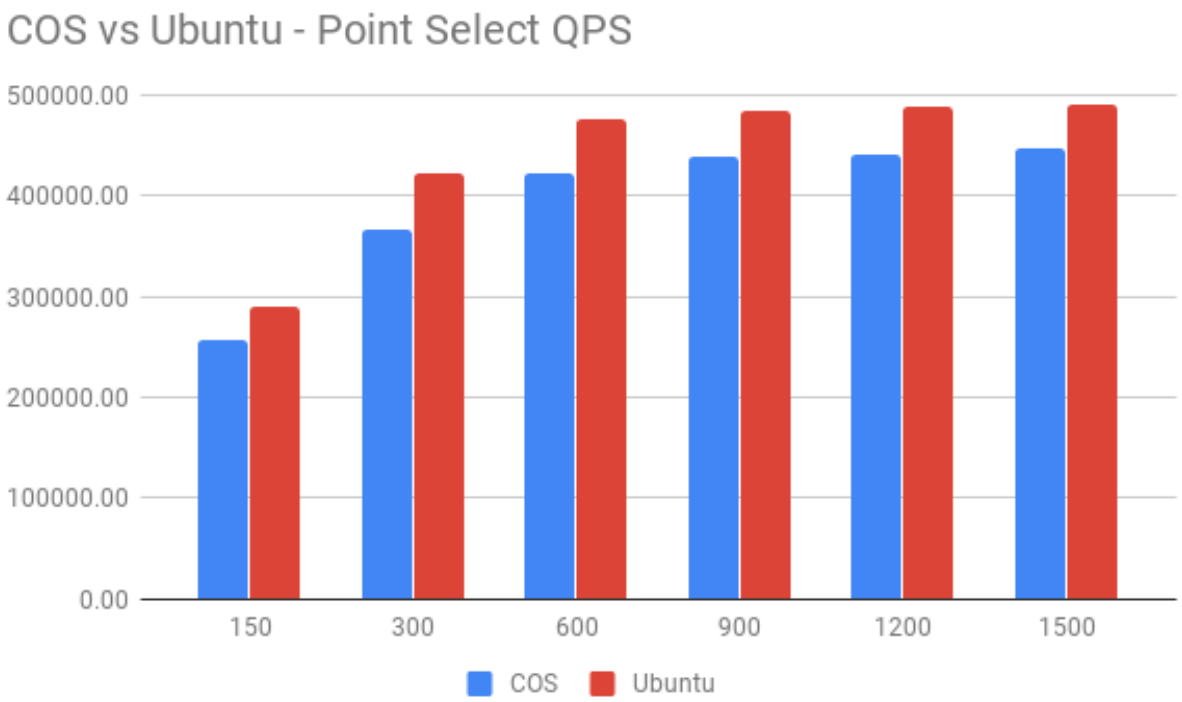


Figure 5: COS vs Ubuntu

Latency **对比：**

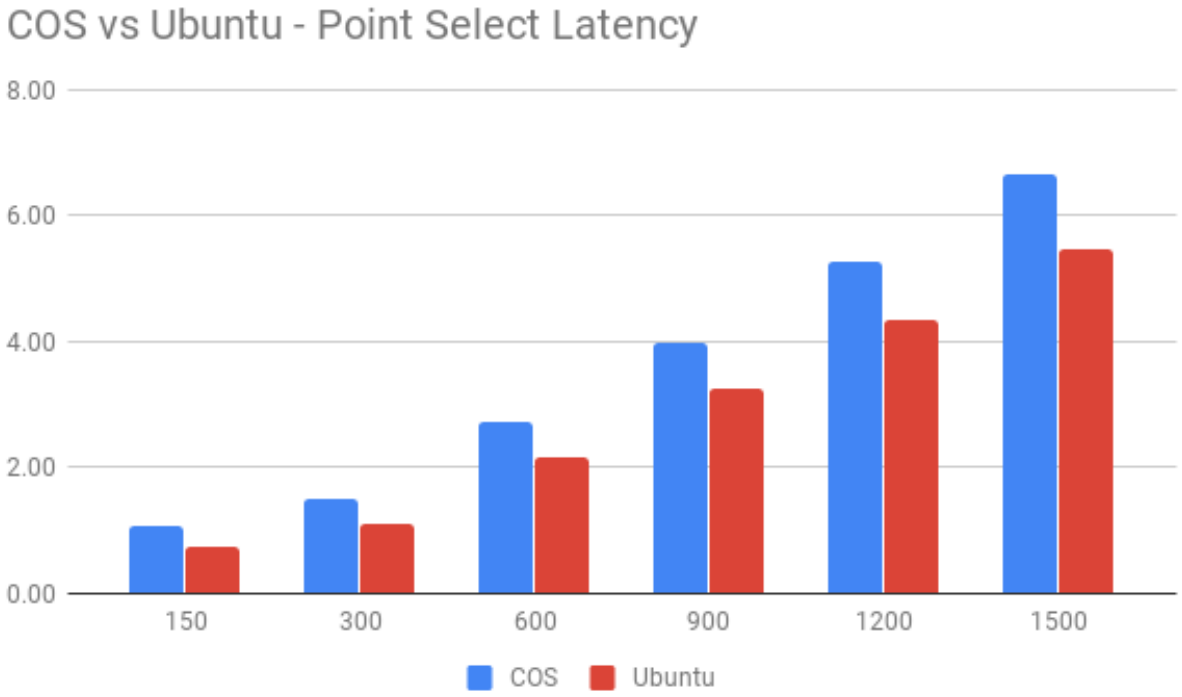


Figure 6: COS vs Ubuntu

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。

3.1.3.1.3 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：

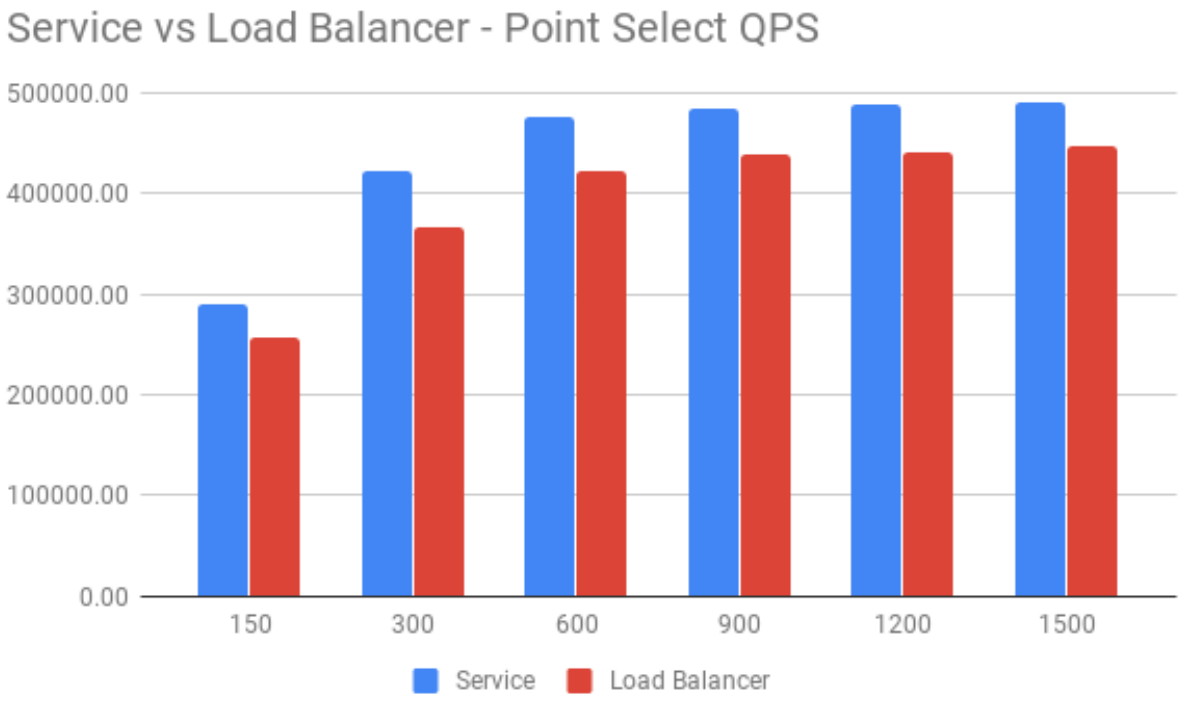


Figure 7: Service vs Load Balancer

Latency 对比：

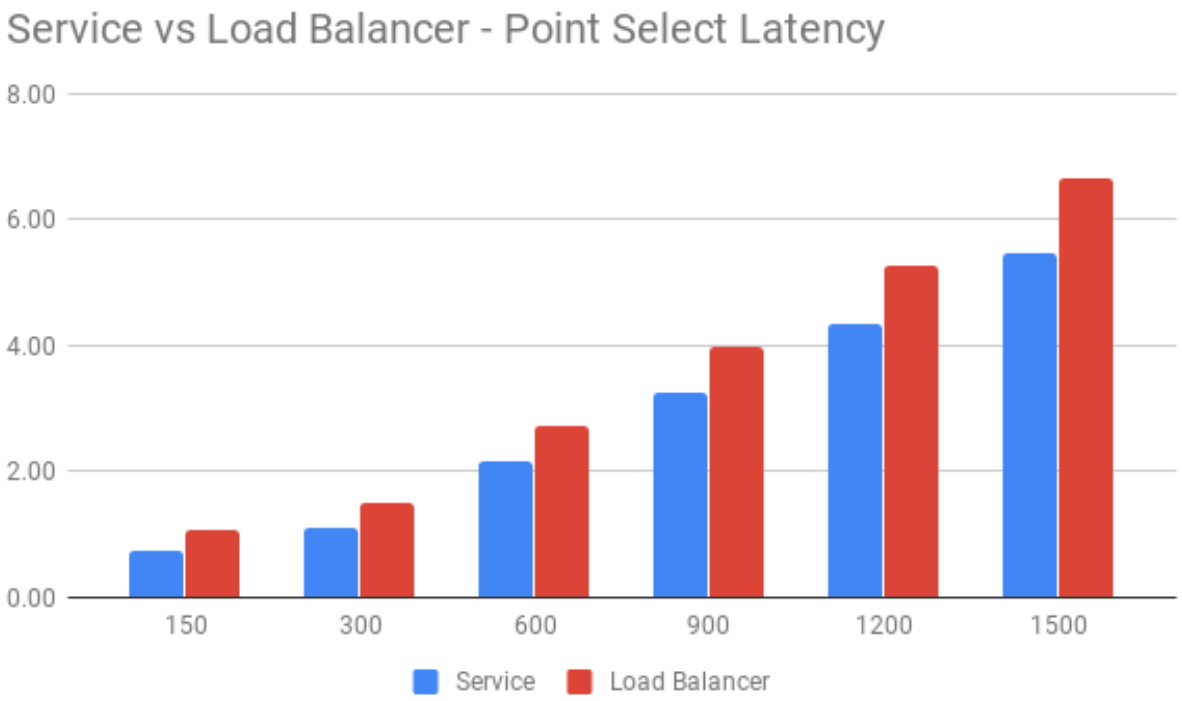


Figure 8: Service vs Load Balancer

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。

3.1.3.1.4 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：**

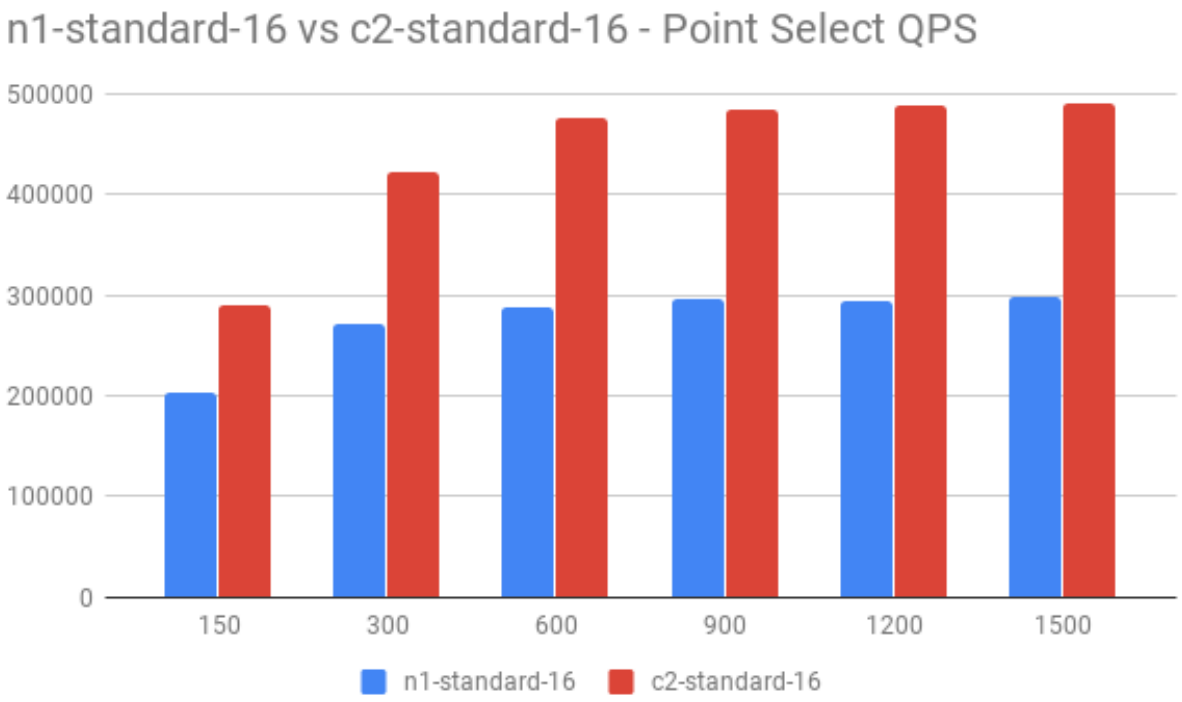


Figure 9: n1-standard-16 vs c2-standard-16

Latency **对比：**

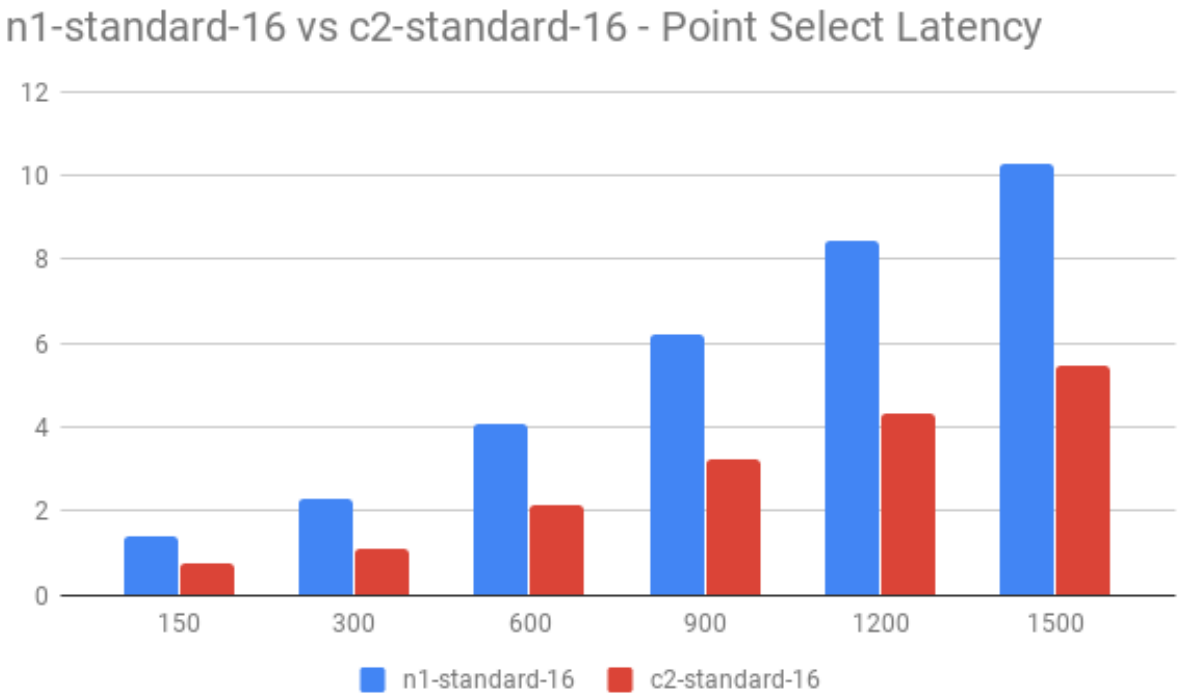


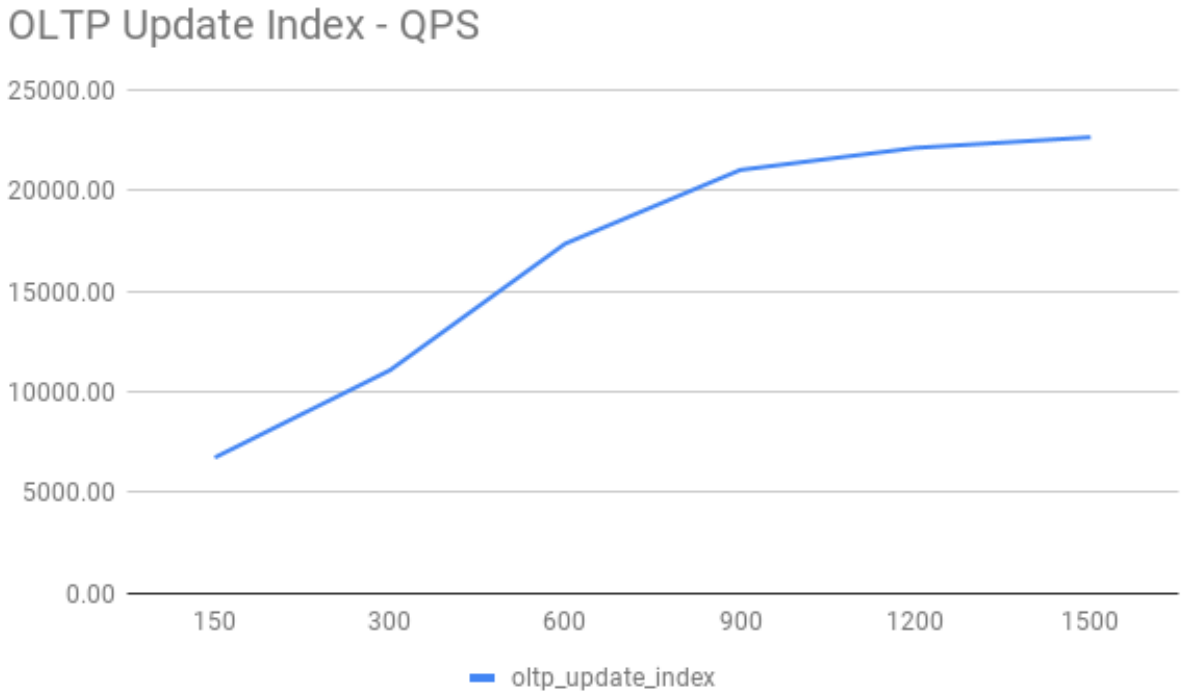
Figure 10: n1-standard-16 vs c2-standard-16

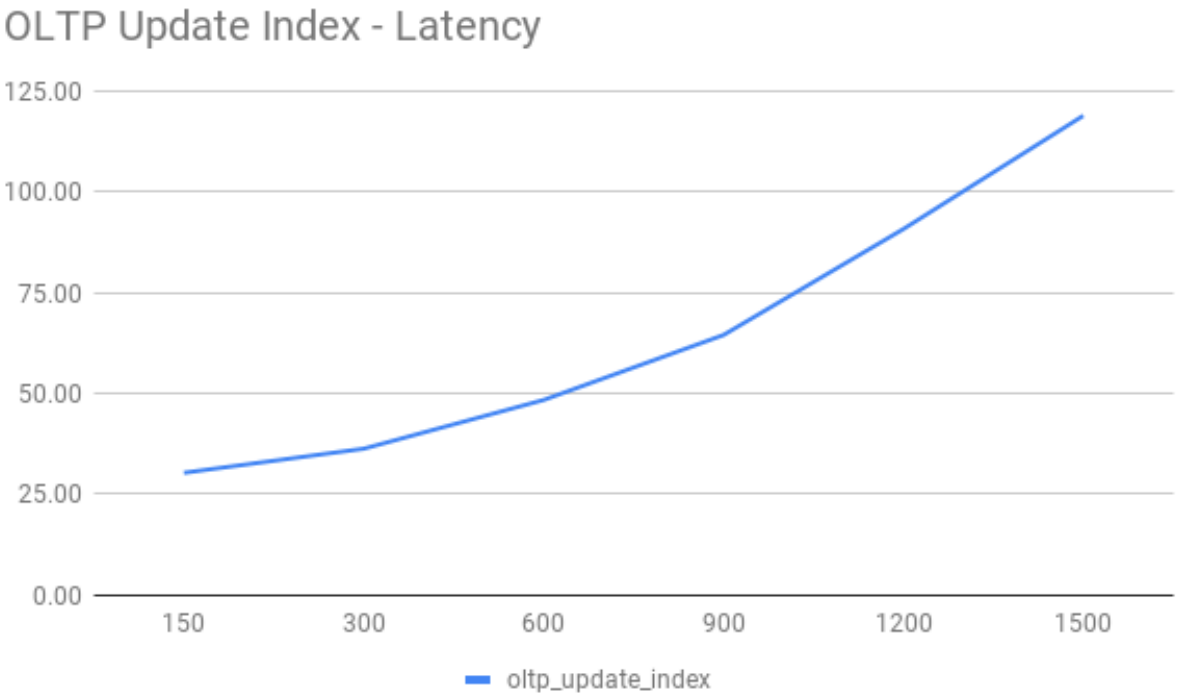
3.1.3.2 OLTP 其他测试

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。

3.1.3.2.1 OLTP Update Index

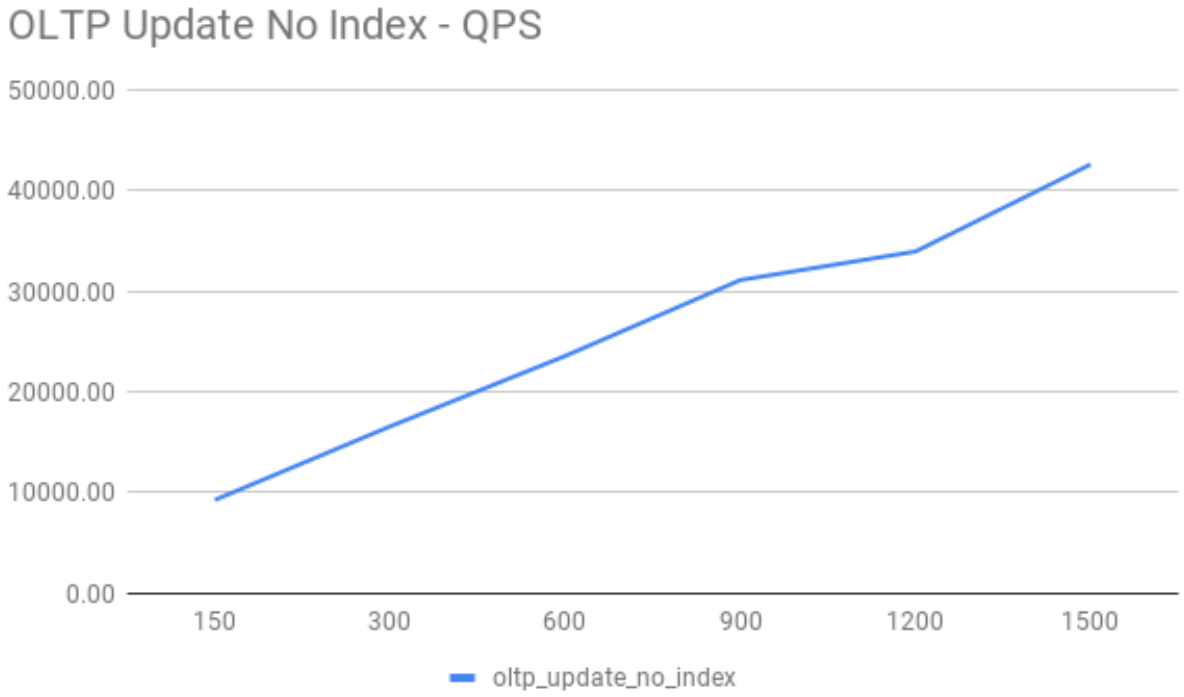
Threads	QPS	95% latency(ms)
150	6726.59	30.26
300	11067.55	36.24
600	17358.46	48.34
900	21025.23	64.47
1200	22121.87	90.78
1500	22650.13	118.92

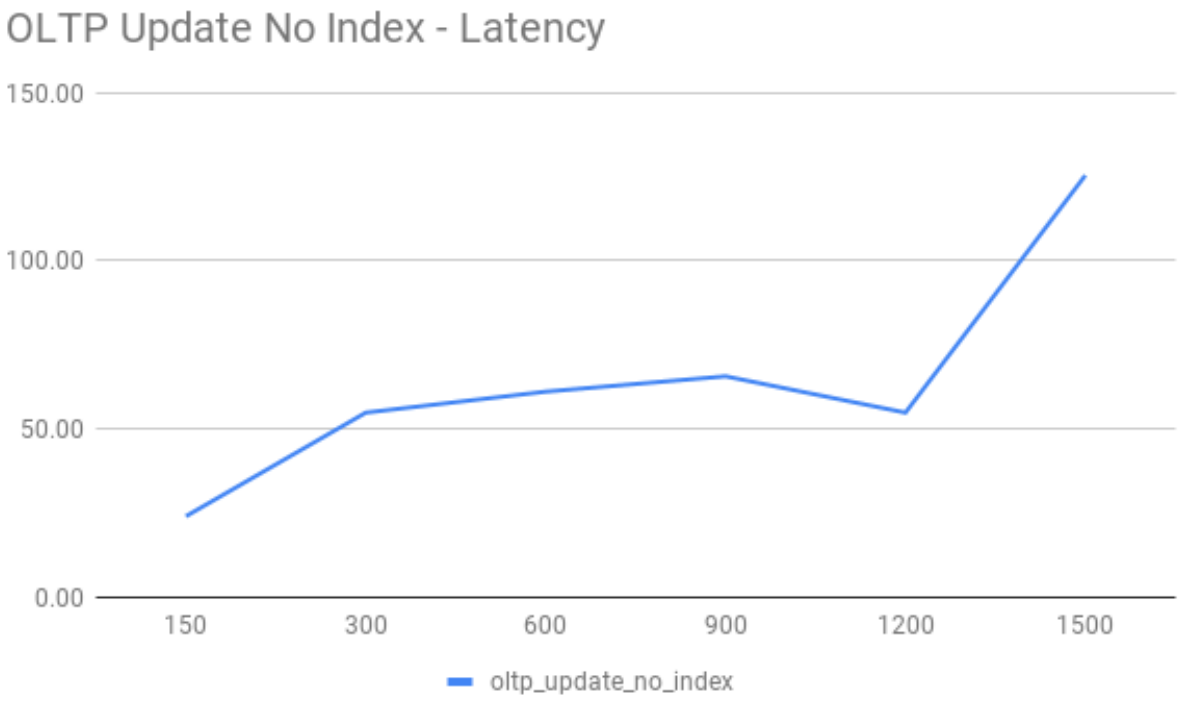




3.1.3.2.2 OLTP Update Non Index

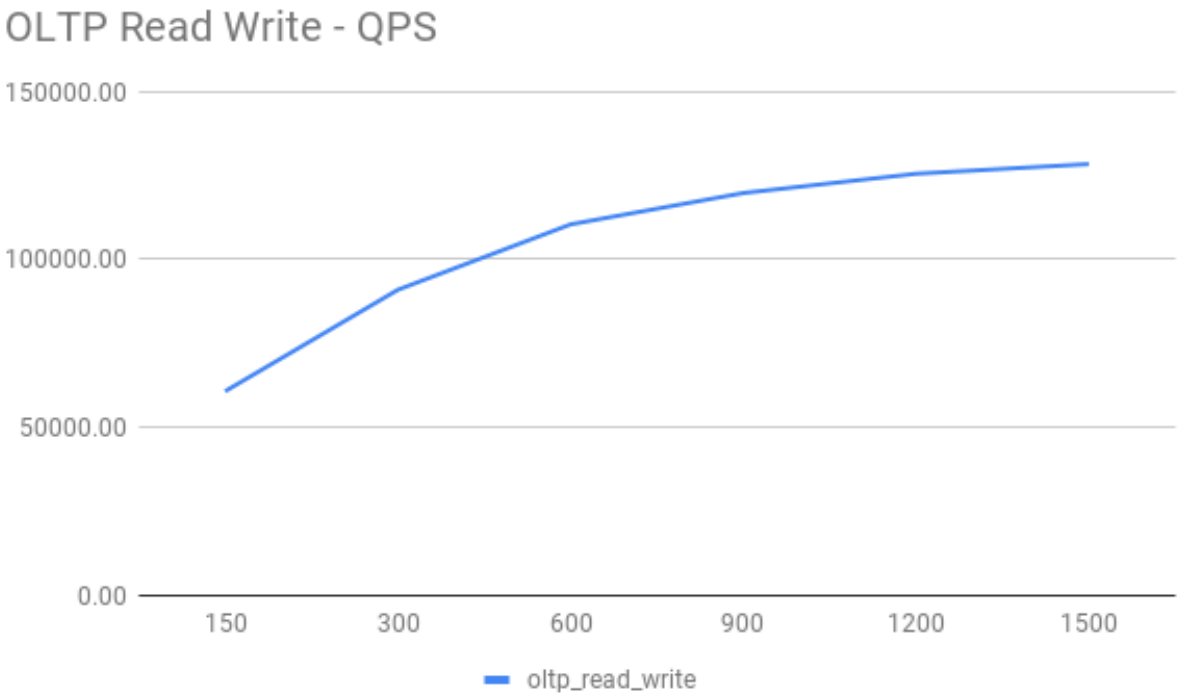
Threads	QPS	95% latency(ms)
150	9230.60	23.95
300	16543.63	54.83
600	23551.01	61.08
900	31100.10	65.65
1200	33942.60	54.83
1500	42603.13	125.52

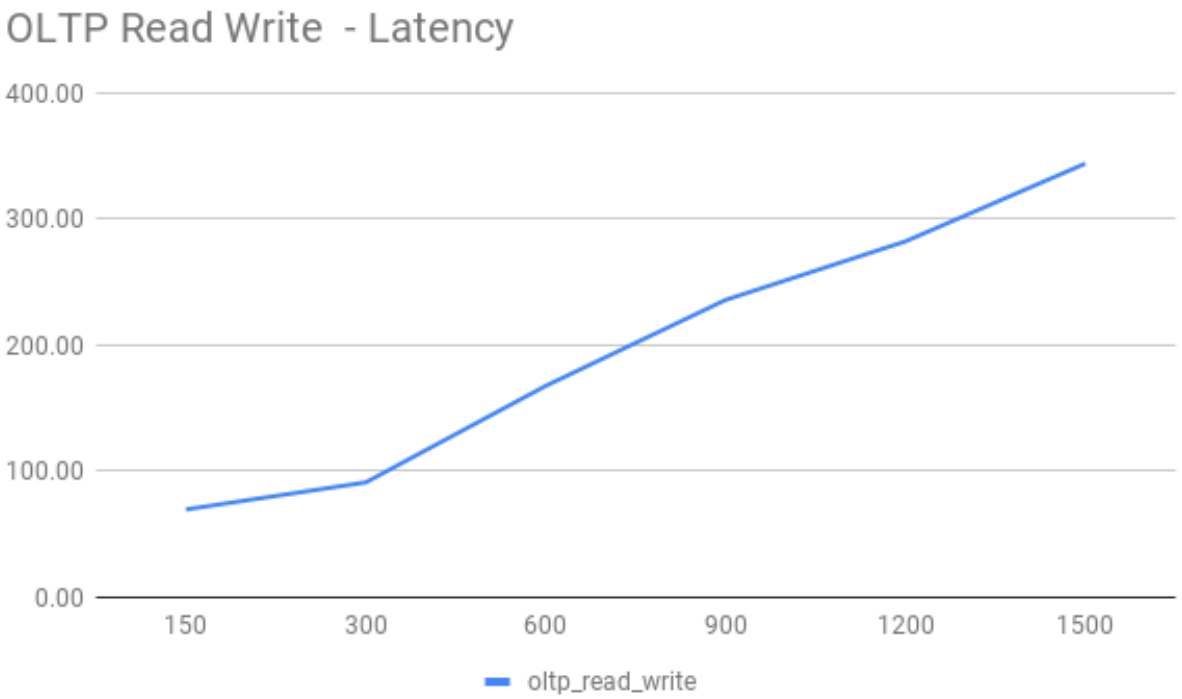




3.1.3.2.3 OLTP Read Write

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





3.1.3.3 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：**

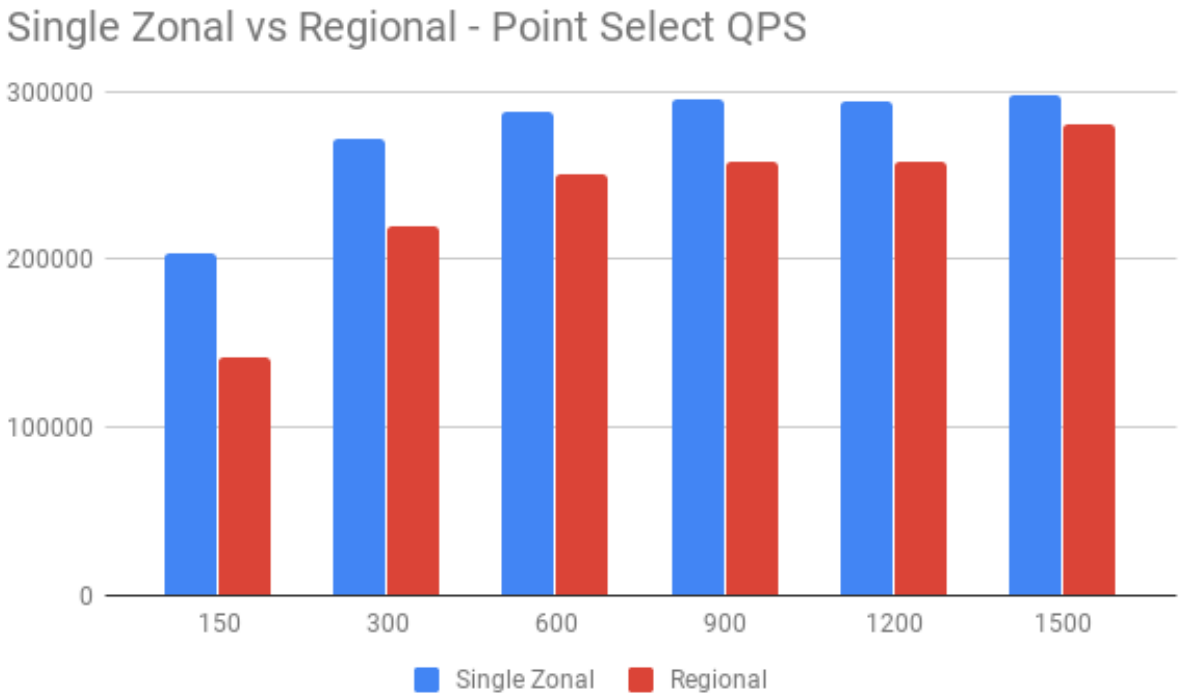


Figure 11: Single Zonal vs Regional

Latency 对比：

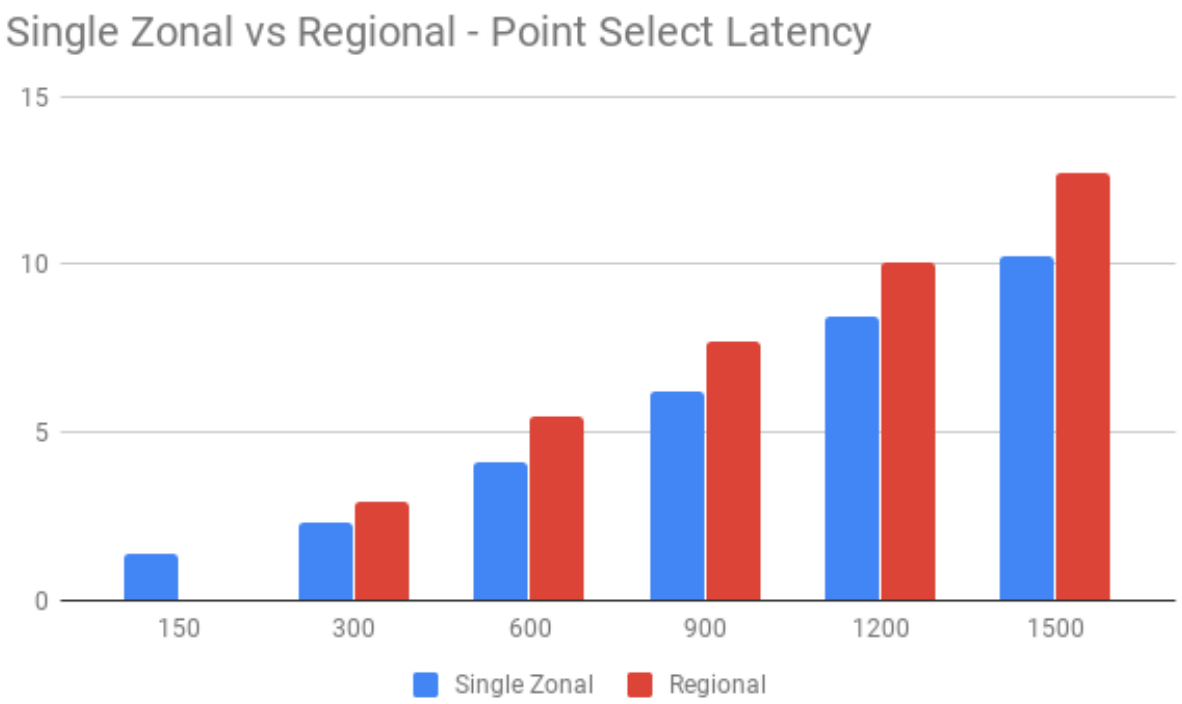


Figure 12: Single Zonal vs Regional

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。

3.1.4 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）。

4 快速上手

4.1 在 GCP 上通过 Kubernetes 部署 TiDB 集群

本文介绍如何使用 TiDB Operator 在 GCP 上部署 TiDB 集群。本教程需要在 Google Cloud Shell 上运行。

所包含的步骤如下：

- 启动一个包含 3 个节点的 Kubernetes 集群（可选）
- 安装 Kubernetes 包管理工具 Helm
- 部署 TiDB Operator
- 部署 TiDB 集群
- 访问 TiDB 集群
- 扩容 TiDB 集群
- 删除 Kubernetes 集群（可选）

> 警告：
>
> 对于生产环境，不要使用此方式进行部署。

4.1.1 选择一个项目

本教程会启动一个包含 3 个 `n1-standard-1` 类型节点的 Kubernetes 集群。价格信息可以参考 All pricing。

继续之前请选择一个项目：

4.1.2 启用 API

本教程需要使用计算和容器 API。继续之前请启用：

4.1.3 配置 gcloud

这一步配置 glcoud 默认访问你要用的项目和可用区，可以简化后面用到的命令：

```
gcloud config set project {{project-id}} && \
gcloud config set compute/zone us-west1-a
```

4.1.4 启动 3 个节点的 Kubernetes 集群

下面命令启动一个包含 3 个 n1-standard-1 类型节点的 Kubernetes 集群。

命令执行需要几分钟时间：

```
gcloud container clusters create tidb
```

集群启动完成后，将其设置为默认集群：

```
gcloud config set container/cluster tidb
```

最后验证 kubectl 可以访问集群并且 3 个节点正常运行：

```
kubectl get nodes
```

如果所有节点状态为 Ready，恭喜你，你已经成功搭建你的第一个 Kubernetes 集群。

4.1.5 安装 Helm

Helm 是 Kubernetes 包管理工具，通过 Helm 可以一键安装 TiDB 的所有分布式组件。安装 Helm 需要同时安装服务端和客户端组件。

安装 helm：

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get | bash
```

复制 helm 到你的 $HOME 目录下，这样即使 Google Cloud Shell 超时断开连接，再次登录后仍然可以访问 Helm：

```
mkdir -p ~/bin && \
cp /usr/local/bin/helm ~/bin && \
echo 'PATH="$PATH:$HOME/bin"' >> ~/.bashrc
```

Helm 正常工作需要一定的权限：

```
kubectl apply -f ./manifests/tiller-rbac.yaml && \
helm init --service-account tiller --upgrade
```

tiller 是 Helm 的服务端组件，初始化完成需要几分钟时间：

```
watch "kubectl get pods --namespace kube-system | grep tiller"
```

当 Pod 状态为 Running，Ctrl+C 停止并继续下一步。

4.1.6 添加 Helm 仓库

PingCAP Helm 仓库中存放着 PingCAP 发布的 charts，例如 tidb-operator、tidb-cluster 和 tidb-backup 等等。使用下面命令添加仓库：

```
helm repo add pingcap https://charts.pingcap.org/ && \
helm repo list
```

然后你可以查看可用的 chart：

```
helm repo update && \
helm search tidb-cluster -l && \
helm search tidb-operator -l
```

4.1.7 部署 TiDB 集群

> 注意：
>
> <chartVersion> 在后面文档中代表 chart 版本，例如 v1.0.0。

第一个要安装的 TiDB 组件是 TiDB Operator，TiDB Operator 是管理组件，结合 Kubernetes 启动 TiDB 集群并保证集群正常运行。执行下面命令之前请确保在 tidb-operator 目录下：

```
kubectl apply -f ./manifests/crd.yaml && \
kubectl apply -f ./manifests/gke/persistent-disk.yaml && \
helm install pingcap/tidb-operator -n tidb-admin --namespace=tidb-admin --
    ↪ version=<chartVersion>
```

可以通过下面命令观察 Operator 启动情况：

```
kubectl get pods --namespace tidb-admin -o wide --watch
```

如果 tidb-scheduler 和 tidb-controller-manager 状态都为 Running，Ctrl+C 停止并继续下一步部署一个 TiDB 集群！

4.1.8 部署你的第一个 TiDB 集群

我们可以一键部署一个 TiDB 集群：

```
helm install pingcap/tidb-cluster -n demo --namespace=tidb --set pd.
    ↪ storageClassName=pd-ssd,tikv.storageClassName=pd-ssd --version=<
    ↪ chartVersion>
```

集群启动需要几分钟时间，可以通过下面命令观察状态：

```
kubectl get pods --namespace tidb -o wide --watch
```

TiDB 集群包含 2 个 TiDB pod，3 个 TiKV pod 和 3 个 PD pod。如果所有 pod 状态都为 Running，Ctrl+C 停止并继续！

4.1.9 访问 TiDB 集群

从 pod 启动、运行到服务可以访问有一些延时，可以通过下面命令查看服务：

```
kubectl get svc -n tidb --watch
```

如果看到 demo-tidb 出现，说明服务已经可以访问，可以 Ctrl+C 停止。

要访问 Kubernetes 集群中的 TiDB 服务，可以在 TiDB 服务和 Google Cloud Shell 之间建立一条隧道。建议这种方式只用于调试，因为如果 Google Cloud Shell 重启，隧道不会自动重新建立。要建立隧道：

```
kubectl -n tidb port-forward svc/demo-tidb 4000:4000 &>/tmp/port-forward.log
    ↪ &
```

在 Cloud Shell 上运行：

```
sudo apt-get install -y mysql-client && \
mysql -h 127.0.0.1 -u root -P 4000
```

在 MySQL 终端中输入一条 MySQL 命令：

```
select tidb_version();
```

如果用 Helm 安装的过程中没有指定密码，现在可以设置：

```
SET PASSWORD FOR 'root'@'%' = '<change-to-your-password>';
```

> 注意：
>
> 这条命令中包含一些特殊字符，Google Cloud Shell 无法自动填充，你需要手动复制、粘贴到控制台中。

恭喜，你已经启动并运行一个兼容 MySQL 的分布式 TiDB 数据库！

4.1.10 扩容 TiDB 集群

我们可以一键扩容 TiDB 集群。如下命令可以扩容 TiKV：

```
helm upgrade demo pingcap/tidb-cluster --set pd.storageClassName=pd-ssd,tikv
    ↪ .storageClassName=pd-ssd,tikv.replicas=5 --version=<chartVersion>
```

TiKV 的 Pod 数量从 3 增加到了 5。可以通过下面命令查看：

```
kubectl get po -n tidb
```

4.1.11 访问 Grafana 面板

要访问 Grafana 面板，可以在 Grafana 服务和 shell 之间建立一条隧道，可以使用如下命令：

```
kubectl -n tidb port-forward svc/demo-grafana 3000:3000 &>/dev/null &
```

在 Cloud Shell 中，点击 Web Preview 按钮并输入端口 3000，将打开一个新的浏览器标签页访问 Grafana 面板。或者也可以在新浏览器标签或者窗口中直接访问 URL：https://ssh.cloud.google.com/devshell/proxy?port=3000。

如果没有使用 Cloud Shell，可以在浏览器中访问 `localhost:3000`。

4.1.12 销毁 TiDB 集群

如果不再需要这个 TiDB 集群，可以使用下面命令删除集群：

```
helm delete demo --purge
```

上面的命令只会删除运行的 Pod，但是数据还会保留。如果你不再需要那些数据，可以执行下面的命令清除数据和动态创建的持久化磁盘：

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

4.1.13 删除 Kubernetes 集群

实验结束后，可以使用如下命令删除 Kubernetes 集群：

```
gcloud container clusters delete tidb
```

4.1.14 更多信息

我们还提供简单的基于 Terraform 的部署方案。

4.2 在 Minikube 集群上部署 TiDB 集群

Minikube 可以让你在个人电脑上的虚拟机中创建一个 Kubernetes 集群，支持 macOS、Linux 和 Windows 系统。本文介绍如何在 Minikube 集群上部署 TiDB 集群。

> 警告：
>
> - 对于生产环境，不要使用此方式进行部署。
> - 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。

4.2.1 安装 Minikube 并启动 Kubernetes 集群

参考安装 Minikube，在你的机器上安装 Minikube 1.0.0+。

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

```
minikube start
```

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` 替换为你自己的 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 通过虚拟机（默认）运行，`127.0.0.1` 是虚拟机本身，有些情况下你可能想要使用你的主机的实际 IP。

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

4.2.2 安装 kubectl 并访问集群

Kubernetes 命令行工具 kubectl，可以让你执行命令访问 Kubernetes 集群。

参考 Install and Set Up kubectl 安装和配置 kubectl。

kubectl 安装完成后，测试 Minikube Kubernetes 集群：

```
kubectl cluster-info
```

4.2.3 安装 TiDB Operator 并运行 TiDB 集群

4.2.3.1 安装 Helm

Helm 是 Kubernetes 包管理工具，通过 Helm 可以一键安装 TiDB 的所有分布式组件。安装 Helm 需要同时安装服务端和客户端组件。

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get | bash
```

安装 helm tiller：

```
helm init
```

如果无法访问 gcr.io，你可以尝试 mirror 仓库：

```
helm init --upgrade --tiller-image registry.cn-hangzhou.aliyuncs.com/
    ↪ google_containers/tiller:$(helm version --client --short | grep -Eo '
    ↪ v[0-9]\.[0-9]+\.[0-9]+')
```

安装完成后，执行 `helm version` 会同时显示客户端和服务端组件版本：

```
helm version
```

输出类似如下内容：

```
Client: &version.Version{SemVer:"v2.13.1",
GitCommit:"618447cbf203d147601b4b9bd7f8c37a5d39fbb4", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.13.1",
GitCommit:"618447cbf203d147601b4b9bd7f8c37a5d39fbb4", GitTreeState:"clean"}
```

如果只显示客户端版本，表示 `helm` 无法连接到服务端。通过 `kubectl` 查看 tiller pod 是否在运行：

```
kubectl -n kube-system get pods -l app=helm
```

4.2.3.2 添加 Helm 仓库

Helm 仓库 (https://charts.pingcap.org/) 存放着 PingCAP 发布的 charts，例如 tidb-operator、tidb-cluster 和 tidb-backup 等等。可使用以下命令添加仓库：

```
helm repo add pingcap https://charts.pingcap.org/ && \
helm repo list
```

然后可以查看可用的 chart：

```
helm repo update && \
helm search tidb-cluster -l && \
helm search tidb-operator -l
```

4.2.3.3 在 Kubernetes 集群上安装 TiDB Operator

> 注意：
>
> <chartVersion> 在后面文档中代表 chart 版本，例如 v1.0.0。

克隆 tidb-operator 代码库并安装 TiDB Operator：

```
git clone --depth=1 https://github.com/pingcap/tidb-operator && \
cd tidb-operator && \
kubectl apply -f ./manifests/crd.yaml && \
helm install pingcap/tidb-operator --name tidb-operator --namespace tidb-
    ↪ admin --version=<chartVersion>
```

然后，可以通过如下命令查看 TiDB Operator 的启动情况：

```
kubectl get pods --namespace tidb-admin -o wide --watch
```

如果无法访问 gcr.io (Pod 由于 ErrImagePull 无法启动)，可以尝试从 mirror 仓库中拉取 kube-scheduler 镜像。可以通过以下命令升级 tidb-operator：

```
helm upgrade tidb-operator pingcap/tidb-operator --namespace tidb-admin --
    ↪ set \
  scheduler.kubeSchedulerImageName=registry.cn-hangzhou.aliyuncs.com/
      ↪ google_containers/kube-scheduler --version=<chartVersion>
```

如果 tidb-scheduler 和 tidb-controller-manager 都进入 running 状态，你可以继续下一步启动一个 TiDB 集群。

4.2.3.4 启动 TiDB 集群

通过下面命令启动 TiDB 集群：

```
helm install pingcap/tidb-cluster --name demo --set \
  schedulerName=default-scheduler,pd.storageClassName=standard,tikv.
    ↪ storageClassName=standard,pd.replicas=1,tikv.replicas=1,tidb.
    ↪ replicas=1 --version=<chartVersion>
```

可以通过下面命令观察集群的状态：

```
kubectl get pods --namespace default -l app.kubernetes.io/instance=demo -o
  ↪ wide --watch
```

通过 Ctrl+C 停止观察。

4.2.4 测试 TiDB 集群

测试 TiDB 集群之前，请确保已经安装 MySQL 客户端。从 Pod 启动、运行到服务可以访问有一些延时，可以通过下面命令查看服务：

```
kubectl get svc --watch
```

如果看到 `demo-tidb` 出现，说明服务已经可以访问，可以 Ctrl+C 停止。

按照以下步骤访问 TiDB 集群：

1. 转发本地端口到 TiDB 端口。

   ```
   kubectl port-forward svc/demo-tidb 4000:4000
   ```

2. 在另一个终端窗口中，通过 MySQL 客户端访问 TiDB：

   ```
   mysql -h 127.0.0.1 -P 4000 -uroot
   ```

   或者可以直接执行 SQL 命令：

   ```
   mysql -h 127.0.0.1 -P 4000 -uroot -e 'select tidb_version();'
   ```

4.2.5 监控 TiDB 集群

按照以下步骤监控 TiDB 集群状态：

1. 转发本地端口到 Grafana 端口。

   ```
   kubectl port-forward svc/demo-grafana 3000:3000
   ```

2. 打开浏览器，通过 http://localhost:3000 访问 Grafana。

   或者，Minikube 提供了 minikube service 的方式暴露 Grafana 服务，可以更方便的接入。

   ```
   minikube service demo-grafana
   ```

   上述命令会自动搭建代理并在浏览器中打开 Grafana。

4.2.5.1 删除 TiDB 集群

通过下面命令删除 demo 集群：

```
helm delete --purge demo
```

更新 demo 集群使用的 PV 的 reclaim 策略为 Delete：

```
kubectl get pv -l app.kubernetes.io/instance=demo -o name | xargs -I {}
    ↪ kubectl patch {} -p '{"spec":{"persistentVolumeReclaimPolicy":"Delete
    ↪ "}}'
```

删除 PVC：

```
kubectl delete pvc -l app.kubernetes.io/managed-by=tidb-operator
```

4.2.6 FAQ

4.2.6.1 Minikube 上的 TiDB 集群不响应或者响应非常慢

Minikube 虚拟机默认配置为 2048 MB 内存和 2 个 CPU。可以在 minikube start 时通过--memory 和 --cpus 选项为其分配更多资源。注意，为了使配置修改生效，你需要重新创建 Minikube 虚拟机。

```
minikube delete && \
minikube start --cpus 4 --memory 4096 ...
```

5 部署

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。

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

5.1.2 内核参数设置

配置项	设置值
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
net.bridge.bridge-nf-call-iptables	1
net.bridge.bridge-nf-call-arptables	1
net.bridge.bridge-nf-call-ip6tables	1

在设置 `net.bridge.bridge-nf-call-*` 这几项参数时，如果选项报错，则可通过如下命令检查是否已经加载该模块：

```
lsmod|grep br_netfilter
```

如果没有加载，则执行如下命令加载：

```
modprobe br_netfilter
```

同时还需要关闭每个部署 Kubernetes 节点的 swap，执行如下命令：

```
swapoff -a
```

执行如下命令检查 swap 是否已经关闭：

```
free -m
```

如果执行命令后输出显示 swap 一列全是 0，则表明 swap 已经关闭。

此外，为了永久性地关闭 swap，还需要将 `/etc/fstab` 中 swap 相关的条目全部删除。

在上述内容都设置完成后，还需要检查是否给机器配置了 SMP IRQ Affinity，也就是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。对于 TiDB 集群来说，网卡处理包的速度对集群的吞吐率影响很大。因此下文主要描述如何将网卡中断号绑定到特定的 CPU 上，充分利用多核的优势来提高集群的吞吐率。首先可以通过以下命令来查看网卡对应的中断号：

```
cat /proc/interrupts|grep <iface-name>|awk '{print $1,$NF}'
```

以上命令输出的第一列是中断号，第二列是设备名称。如果是多队列网卡，上面的命令会显示多行信息，网卡的每个队列对应一个中断号。通过以下命令可以查看该中断号被绑定到哪个 CPU 上：

```
cat /proc/irq/<ir_num>/smp_affinity
```

上面命令输出 CPU 序号对应的十六进制值。输出结果欠直观。具体计算方法可参见 SMP IRQ Affinity 文档。

```
cat /proc/irq/<ir_num>/smp_affinity_list
```

上面命令输出 CPU 序号对应的十进制值，输出结果较为直观。

如果多队列网卡对应的所有中断号都已被绑定到不同的 CPU 上，那么该机器的 SMP IRQ Affinity 配置是正确的。如果中断号都落在同一个 CPU 上，则需要进行调整。调整的方式有以下两种：

- 方法一：开启 irqbalance 服务。在 centos7 系统上的开启命令如下:

  ```
  systemctl start irqbalance
  ```

- 方法二：禁用 irqbalance，自定义中断号和 CPU 的绑定关系。详情参见脚本 set_irq_affinity.sh。

上文所描述的是处理多队列网卡和多核心的场景。单队列网卡和多核的场景则有不同的处理方式。在这种场景下，可以使用 RPS/RFS 在软件层面模拟实现硬件的网卡多队列功能 (RSS)。此时不能使用方法一所述的 irqbalance 服务，而是通过使用方法二提供的脚本来设置 RPS。RFS 的配置可以参考这里。

5.1.3 硬件和部署要求

与使用 binary 方式部署 TiDB 集群一致，要求选用 Intel x86-64 架构的 64 位通用硬件服务器，使用万兆网卡。关于 TiDB 集群在物理机上的具体部署需求，参考 TiDB 软件和硬件环境建议配置。

对于服务器 disk、memory、CPU 的选择要根据对集群的容量规划以及部署拓扑来定。线上 Kubernetes 集群部署为了保证高可用，一般需要部署三个 master 节点、三个 etcd 节点以及若干个 worker 节点。同时，为了充分利用机器资源，master 节点一般也充当 worker 节点 (也就是 master 节点上也可以调度负载)。通过 kubelet 设置预留资源来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。

下面按 3 Kubernetes master + 3 etcd + 若干 worker 节点部署方案进行分析。Kubernetes 的多 master 节点高可用部署可参考官方文档。

5.1.4 Kubernetes 系统资源要求

- 每台机器需要一块比较大的 SAS 盘（至少 1T），这块盘用来存 Docker 和 kubelet 的数据目录。Docker 的数据主要包括镜像和容器日志数据，kubelet 主要占盘的数据是 emptyDir 所使用的数据。
- 如果需要部署 Kubernetes 集群的监控系统，且监控数据需要落盘，则也需要考虑为 Prometheus 准备一块 SAS 盘，后面日志监控系统也需要大的 SAS 盘，同时考虑到机器采购最好是同构的这一因素，因此每台机器最好有两块大的 SAS 盘。

> 注意：
> 生产环境建议给这两种类型的盘做 RAID5，至于使用多少块来做 RAID5 可自己决定。

- etcd 的分布建议是和 Kubernetes master 节点保持一致，即有多少个 master 节点就部署多少个 etcd 节点。etcd 数据建议使用 SSD 盘存放。

5.1.5 TiDB 集群资源需求

TiDB 集群由 PD、TiKV、TiDB 三个组件组成，在做容量规划的时候一般按照可以支持多少套 TiDB 集群来算。这里按照标准的 TiDB 集群（3 个 PD + 3 个 TiKV + 2 个 TiDB）来算，下面是对每个组件规划的一种建议：

- PD 组件：PD 占用资源较少，这种集群规模下分配 2C 4GB 即可，占用少量本地盘。

  为了便于管理，可以将所有集群的 PD 都放在 master 节点，比如需要支持 5 套 TiDB 集群，则可以规划 3 个 master 节点，每个节点支持部署 5 个 PD 实例，5 个 PD 实例使用同一块 SSD 盘即可（两三百 GB 的盘即可）。通过 bind mount 的方式在这块 SSD 上创建 5 个目录作为挂载点，操作方式见 Sharing a disk filesystem by multiple filesystem PVs。

  如果后续要添加更多机器支持更多的 TiDB 集群，可以在 master 上用这种方式继续增加 PD 实例。如果 master 上资源耗尽，可以找其它的 worker 节点机器用同样的方式添加 PD 实例。这种方式的好处就是方便规划和管理 PD 实例，坏处就是由于 PD 实例过于集中，这些机器中如果有两台宕机会导致所有的 TiDB 集群不可用。

  因此建议从所有集群里面的机器都拿出一块 SSD 盘像 master 节点一样提供 PD 实例。比如总共 7 台机器，要支持 7 套 TiDB 标准集群的情况下，则需要每台机器上都能支持部署 3 个 PD 实例，如果后续有集群需要通过扩容机器增加容量，也只需要在新的机器上创建 PD 实例。

- TiKV 组件：因为 TiKV 组件的性能很依赖磁盘 I/O 且数据量一般较大，因此建议每个 TiKV 实例独占一块 NVMe 的盘，资源配置为 8C 32GB。如果想要在一个机器上支持部署多个 TiKV 实例，则建议参考这些参数去选择合适的机器，同时在规划容量的时候应当预留出足够的 buffer。

- TiDB 组件：TiDB 组件因为不占用磁盘，因此在规划的时候只需要考虑其占用的 CPU 和内存资源即可，这里也按 8C 32 GB 的容量来计算。

5.1.6 TiDB 集群规划示例

通过上面的分析，这里给出一个支持部署 5 套规模为 3 个 PD + 3 个 TiKV + 2 个 TiDB 集群的例子，其中 PD 配置为 2C 4GB，TiDB 配置为 8C 32GB，TiKV 配置为 8C 32GB。Kubernetes 节点有 7 个，其中有 3 个节点既是 master 又是 worker 节点，另外 4 个是纯 worker 节点。各节点上部署组件情况如下：

- 每台 master 节点：
  - 1 etcd (2C 4GB) + 2 PD (2 * 2C 2 * 4GB) + 3 TiKV (3 * 8C 3 * 32GB) + 1 TiDB (8C 32GB)，总共是 38C 140GB
  - 两块 SSD 盘，一块给 etcd，另外一块给 2 个 PD 实例
  - 做了 RAID5 的 SAS 盘，给 Docker 和 kubelet 做数据盘
  - 三块 NVMe 盘给 TiKV 实例
- 每台 worker 节点：
  - 3 PD (3 * 2C 3 * 4GB) + 2 TiKV (2 * 8C 2 * 32GB) + 2 TiDB (2 * 8C 2 * 32GB)，总共是 38C 140GB
  - 一块 SSD 盘给三个 PD 实例
  - 做了 RAID5 的 SAS 盘，给 Docker 和 kubelet 做数据盘
  - 两块 NVMe 盘给 TiKV 实例

从上面的分析来看，要支持 5 套 TiDB 集群容量共需要 7 台物理机，其中 3 台为 master 兼 worker 节点，其余 4 台为 worker 节点，机器配置需求如下：

- master 兼 worker 节点：48C 192GB；2 块 SSD 盘，一块做了 RAID5 的 SAS 盘，三块 NVMe 盘
- worker 节点：48C 192GB；1 块 SSD 盘，一块做了 RAID5 的 SAS 盘，两块 NVMe 盘

使用上面的机器配置，除去各个组件占用的资源外，还有比较多的预留资源。如果要考虑加监控和日志组件，则可以用同样的方法去规划需要采购的机器类型以及配置。

另外，在生产环境的使用上尽量不要在 master 节点部署 TiDB 实例，或者尽可能少地部署 TiDB 实例。这里的主要考虑点是网卡带宽，因为 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。

5.2 在 Kubernetes 上部署 TiDB Operator

本文介绍如何在 Kubernetes 上部署 TiDB Operator。

5.2.1 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 版本 >= 2.11.0 && < 2.16.4

> 注意：
>
> - 尽管 TiDB Operator 可以使用网络卷持久化 TiDB 数据，TiDB 数据自身会存多副本，再走额外的网络卷性能会受到很大影响。强烈建议搭建本地卷以提高性能。
> - 跨多可用区的网络卷需要 Kubernetes v1.12 或者更高版本。在 `tidb-backup` chart 配置中，建议使用网络卷存储备份数据。

5.2.2 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。如果你使用 AWS、GKE 或者本机，下面是快速上手教程：

- Google GKE 教程
- AWS EKS 教程

如果你要使用不同环境，必须在 Kubernetes 集群中安装 DNS 插件。可以根据官方文档搭建 DNS 插件。

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。可以根据这一步自动配置本地持久化卷。

Kubernetes 集群建议启用 RBAC。否则，需要在 `tidb-operator` 和 `tidb-cluster` chart 的 `values.yaml` 中设置 `rbac.create` 为 `false`。

TiDB 默认会使用很多文件描述符，工作节点和上面的 Docker 进程的 `ulimit` 必须设置大于等于 `1048576`：

- 设置工作节点的 `ulimit` 值，详情可以参考如何设置 ulimit

```
sudo vim /etc/security/limits.conf
```

设置 root 账号的 soft 和 hard 的 nofile 大于等于 1048576

- 设置 Docker 服务的 ulimit

```
sudo vim /etc/systemd/system/docker.service
```

设置 LimitNOFILE 大于等于 1048576。

> **注意：**
>
> LimitNOFILE 需要显式设置为 1048576 或者更大，而不是默认的 infinity，由于 systemd 的 bug，infinity 在 systemd 某些版本中指的是 65536。

5.2.3 安装 Helm

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

5.2.4 配置本地持久化卷

5.2.4.1 准备本地卷

参考本地 PV 配置在你的 Kubernetes 集群中配置本地持久化卷。

5.2.5 安装 TiDB Operator

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/
    ↪ master/manifests/crd.yaml && \
kubectl get crd tidbclusters.pingcap.com
```

创建 TidbCluster 自定义资源类型后，接下来在 Kubernetes 集群上安装 TiDB Operator。

1. 获取你要安装的 tidb-operator chart 中的 values.yaml 文件：

```
mkdir -p /home/tidb/tidb-operator && \
helm inspect values pingcap/tidb-operator --version=<chart-version> > /
    ↪ home/tidb/tidb-operator/values-tidb-operator.yaml
```

> **注意：**
>
> <chart-version> 在后续文档中代表 chart 版本，例如 v1.0.0，可以通过 helm search -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，如果下载不了该镜像，可以修改 /home/tidb/tidb-operator/values-tidb-operator.yaml 文件中的 scheduler.kubeSchedulerImageName 为 registry.cn-hangzhou.aliyuncs.com/google_containers/kube-scheduler。

3. 安装 TiDB Operator

```
helm install pingcap/tidb-operator --name=tidb-operator --namespace=
    ↪ tidb-admin --version=<chart-version> -f /home/tidb/tidb-operator/
    ↪ values-tidb-operator.yaml && \
kubectl get po -n tidb-admin -l app.kubernetes.io/name=tidb-operator
```

5.2.6 自定义 TiDB Operator

通过修改 /home/tidb/tidb-operator/values-tidb-operator.yaml 中的配置自定义 TiDB Operator。后续文档使用 values.yaml 指代 /home/tidb/tidb-operator/values-tidb-operator.yaml。

TiDB Operator 有两个组件：

- tidb-controller-manager
- tidb-scheduler

这两个组件是无状态的，通过 Deployment 部署。你可以在 values.yaml 中自定义资源 limit、request 和 replicas。

修改为 values.yaml 后，执行下面命令使配置生效：

```
helm upgrade tidb-operator pingcap/tidb-operator --version=<chart-version> -
    ↪ f /home/tidb/tidb-operator/values-tidb-operator.yaml
```

5.3 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。

5.3.1 前置条件

- 参考TiDB Operator 完成集群中的 TiDB Operator 部署；
- 参考使用 Helm 安装 Helm 并配置 PingCAP 官方 chart 仓库。

5.3.2 配置 TiDB 集群

通过下面命令获取待安装的 tidb-cluster chart 的 values.yaml 配置文件：

```
mkdir -p /home/tidb/<release-name> && \
helm inspect values pingcap/tidb-cluster --version=<chart-version> > /home/
    ↪ tidb/<release-name>/values-<release-name>.yaml
```

注意：

- /home/tidb 可以替换为你想用的目录。
- release-name 将会作为 Kubernetes 相关资源（例如 Pod，Service 等）的前缀名，可以起一个方便记忆的名字，要求全局唯一，通过 helm ls -q 可以查看集群中已经有的 release-name。
- chart-version 是 tidb-cluster chart 发布的版本，可以通过 helm search -l tidb-cluster 查看当前支持的版本。
- 下文会用 values.yaml 指代 /home/tidb/<release-name>/values-<release-name>.yaml。

5.3.2.1 存储类型

集群默认使用 local-storage 存储类型。

- 生产环境：推荐使用本地存储，但实际 Kubernetes 集群中本地存储可能按磁盘类型进行了分类，例如 nvme-disks，sas-disks。
- 演示环境或功能性验证：可以使用网络存储，例如 ebs，nfs 等。

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为 TiDB 集群各组件选择合适的存储类型，通过修改 values.yaml 中各组件的 storageClassName 字段设置存储类型。关于 Kubernetes 集群支持哪些存储类型，请联系系统管理员确定。

注意：

如果创建集群时设置了集群中不存在的存储类型，则会导致集群创建处于 Pending 状态，需要将集群彻底销毁掉。

5.3.2.2 集群拓扑

默认部署的集群拓扑是：3 个 PD Pod，3 个 TiKV Pod，2 个 TiDB Pod 和 1 个监控 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。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 和 TiKV Pod 个数都减小到 1 个，或者将 `values.yaml` 中 `schedulerName` 改为 Kubernetes 内置调度器 `default-scheduler`。

> **警告：**
>
> `default-scheduler` 仅适用于演示环境，改为 `default-scheduler` 后，TiDB 集群的调度将无法保证数据高可用，另外一些其它特性也无法支持，例如 TiDB Pod StableScheduling 等。

其它更多配置参数请参考TiDB 集群部署配置文档。

5.3.3 部署 TiDB 集群

TiDB Operator 部署并配置完成后，可以通过下面命令部署 TiDB 集群：

```
helm install pingcap/tidb-cluster --name=<release-name> --namespace=<
    ↪ namespace> --version=<chart-version> -f /home/tidb/<release-name>/
    ↪ values-<release-name>.yaml
```

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

通过下面命令可以查看 Pod 状态：

```
kubectl get po -n <namespace> -l app.kubernetes.io/instance=<release-name>
```

单个 Kubernetes 集群中可以利用 TiDB Operator 部署管理多套 TiDB 集群，重复以上命令并将 `release-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。

5.4 在 AWS EKS 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 AWS EKS (Elastic Kubernetes Service) 上部署 TiDB 集群。

5.4.1 环境配置准备

部署前，请确认已安装以下软件并完成配置：

- awscli >= 1.16.73，控制 AWS 资源

  要与 AWS 交互，必须配置 awscli。最快的方式是使用 `aws configure` 命令:

```
aws configure
```

  替换下面的 AWS Access Key ID 和 AWS Secret Access Key：

```
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

> **注意：**
> Access key 必须至少具有以下权限：创建 VPC、创建 EBS、创建 EC2 和创建 Role。

- terraform >= 0.12
- kubectl >= 1.12
- helm >= 2.11.0 且 < 2.16.4
- jq
- aws-iam-authenticator，AWS 权限鉴定工具，确保安装在 `PATH` 路径下。

  最简单的安装方法是下载编译好的二进制文件 `aws-iam-authenticator`，如下所示。

  Linux 用户下载二进制文件：

```
curl -o aws-iam-authenticator https://amazon-eks.s3-us-west-2.amazonaws
   ↪ .com/1.12.7/2019-03-27/bin/linux/amd64/aws-iam-authenticator
```

  macOS 用户下载二进制文件：

```
curl -o aws-iam-authenticator https://amazon-eks.s3-us-west-2.amazonaws
   ↪ .com/1.12.7/2019-03-27/bin/darwin/amd64/aws-iam-authenticator
```

二进制文件下载完成后，执行以下操作：

```
chmod +x ./aws-iam-authenticator && \
sudo mv ./aws-iam-authenticator /usr/local/bin/aws-iam-authenticator
```

5.4.2 部署集群

默认部署会创建一个新的 VPC、一个 t2.micro 实例作为堡垒机，并包含以下 ec2 实例作为工作节点的 EKS 集群：

- 3 台 m5.xlarge 实例，部署 PD
- 3 台 c5d.4xlarge 实例，部署 TiKV
- 2 台 c5.4xlarge 实例，部署 TiDB
- 1 台 c5.2xlarge 实例，部署监控组件

使用如下命令部署集群。

从 Github 克隆代码并进入指定路径：

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

使用 terraform 命令初始化并部署集群：

```
terraform init
```

```
terraform apply
```

> 注意：
>
> terraform apply 过程中必须输入 “yes” 才能继续。

整个过程可能至少需要 10 分钟。terraform apply 执行成功后，控制台会输出类似如下的信息：

```
Apply complete! Resources: 67 added, 0 changed, 0 destroyed.

Outputs:

bastion_ip = [
  "34.219.204.217",
]
```

```
default-cluster_monitor-dns = a82db513ba84511e9af170283460e413-1838961480.us
    ↪ -west-2.elb.amazonaws.com
default-cluster_tidb-dns = a82df6d13a84511e9af170283460e413-d3ce3b9335901d8c
    ↪ .elb.us-west-2.amazonaws.com
eks_endpoint = https://9A9A5ABB8303DDD35C0C2835A1801723.yl4.us-west-2.eks.
    ↪ amazonaws.com
eks_version = 1.12
kubeconfig_filename = credentials/kubeconfig_my-cluster
region = us-west-21
```

你可以通过 `terraform output` 命令再次获取上面的输出信息。

> **注意：**
>
> 1.14 版本以前的 EKS 不支持自动开启 NLB 跨可用区负载均衡，因此默认配置下会出现各台 TiDB 实例压力不均衡额状况。生产环境下，强烈建议参考 AWS 官方文档手动开启 NLB 的跨可用区负载均衡。

5.4.3 访问数据库

`terraform apply` 完成后，可先通过 `ssh` 远程连接到堡垒机，再通过 MySQL client 来访问 TiDB 集群。

所需命令如下（用上面的输出信息替换 <> 部分内容）：

```
ssh -i credentials/<eks_name>.pem centos@<bastion_ip>
```

```
mysql -h <tidb_dns> -P 4000 -u root
```

`eks_name` 默认为 `my-cluster`。如果 DNS 名字无法解析，请耐心等待几分钟。

你还可以通过 `kubectl` 和 `helm` 命令使用 kubeconfig 文件 `credentials/kubeconfig_<eks_name>` 和 EKS 集群交互，主要有两种方式，如下所示。

- 指定 –kubeconfig 参数：

```
kubectl --kubeconfig credentials/kubeconfig_<eks_name> get po -n <
    ↪ default_cluster_name>
```

```
helm --kubeconfig credentials/kubeconfig_<eks_name> ls
```

- 或者，设置 KUBECONFIG 环境变量：

```
export KUBECONFIG=$PWD/credentials/kubeconfig_<eks_name>
```

```
kubectl get po -n <default_cluster_name>
```

```
helm ls
```

5.4.4 Grafana 监控

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

Grafana 默认登录信息：

- 用户名：admin
- 密码：admin

5.4.5 升级 TiDB 集群

要升级 TiDB 集群，可编辑 variables.tf 文件，修改 default_cluster_version 变量到更高版本，然后运行 terraform apply。

例如，要升级 TiDB 集群到 3.0.1，则修改 default_cluster_version 为 v3.0.1：

```
variable "default_cluster_version" {
  default = "v3.0.1"
}
```

> 注意：
>
> 升级过程会持续一段时间，你可以通过 kubectl --kubeconfig credentials/kubeconfig_<eks_name> get po -n <default_cluster_name> --watch 命令持续观察升级进度。

5.4.6 扩容 TiDB 集群

若要扩容 TiDB 集群，可按需修改 variables.tf 文件中的 default_cluster_tikv_count 或者 default_cluster_tidb_count 变量，然后运行 terraform apply。

例如，可以将 default_cluster_tidb_count 从 2 改为 4 以扩容 TiDB：

```
variable "default_cluster_tidb_count" {
  default = 4
}
```

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_<eks_name> get po -n <default_cluster_name> --watch` 命令持续观察进度。

5.4.7 自定义

你可以按需修改 `variables.tf` 文件中的默认值，例如集群名称和镜像版本等。

5.4.7.1 自定义 AWS 相关的资源

默认情况下 terraform 脚本会新建 VPC。你也可以通过设置 `create_vpc` 为 `false`，并指定 `vpc_id`、`private_subnet_ids` 和 `public_subnet_ids` 变量为已有的 VPC id、subnet ids 来使用现有的网络。

注意：

- 由于 AWS 和 Terraform 的限制，还不支持复用已有 EKS 集群的 VPC 和 subnets，所以请确保只在你手动创建 VPC 的情况下修改该参数；
- EKS Node 上的 CNI 插件会为每个节点预留一部分 IP 资源，因此 IP 消耗较大，在手动创建 VPC 时，建议将每个 subnet 的掩码长度设置在 18~20 以确保 IP 资源充足，或参考 EKS CNI 插件文档将节点预留的 IP 资源数调低。

由于 TiDB 服务通过 Internal Elastic Load Balancer 暴露，默认情况下，会创建一个 Amazon EC2 实例作为堡垒机，访问创建的 TiDB 集群。堡垒机上预装了 MySQL 和 Sysbench，所以你可以通过 SSH 方式登陆到堡垒机后通过 ELB 访问 TiDB。如果你的 VPC 中已经有了类似的 EC2 实例，你可以通过设置 `create_bastion` 为 `false` 禁掉堡垒机的创建。

TiDB 版本和组件数量也可以在 `variables.tf` 中修改，你可以按照自己的需求配置。

目前，由于 PD 和 TiKV 依赖 NVMe SSD 实例存储卷，TiDB 集群组件的实例类型不能修改。

5.4.7.2 自定义 TiDB 参数配置

Terraform 脚本中为运行在 EKS 上的 TiDB 集群提供了合理的默认配置。有自定义需求时，你可以在 `clusters.tf` 中通过 `override_values` 参数为每个 TiDB 集群指定一个 `values.yaml` 文件来自定义集群参数配置。

作为例子，默认集群使用了 `./default-cluster.yaml` 作为 `values.yaml` 配置文件，并在配置中打开了 "配置文件滚动更新" 特性。

值得注意的是，在 EKS 上部分配置项无法在 `values.yaml` 中进行修改，包括集群版本、副本数、`NodeSelector` 以及 `Tolerations`。`NodeSelector` 和 `Tolerations` 由 Terraform 直接管理以确保基础设施与 TiDB 集群之间的一致性。集群版本和副本数可以通过 `cluster.tf` 文件中的 `tidb-cluster` module 参数来修改。

> **注意：**
>
> 自定义 `values.yaml` 配置文件中，不建议包含如下配置（`tidb-cluster` module 默认固定配置）：

```
pd:
  storageClassName: ebs-gp2
tikv:
  stroageClassName: local-storage
tidb:
  service:
    type: LoadBalancer
    annotations:
      service.beta.kubernetes.io/aws-load-balancer-internal: '0.0.0.0/0'
      service.beta.kubernetes.io/aws-load-balancer-type: nlb
      service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing
          ↪ -enabled: >'true'
  separateSlowLog: true
monitor:
  storage: 100Gi
  storageClassName: ebs-gp2
  persistent: true
  grafana:
    config:
      GF_AUTH_ANONYMOUS_ENABLED: "true"
    service:
      type: LoadBalancer
```

5.4.7.3 自定义 TiDB Operator

你可以通过 variables.tf 中的 operator_values 参数传入自定义的 values.yaml 内容来配置 TiDB Operator。示例如下：

```
variable "operator_values" {
  description = "The helm values file for TiDB Operator, path is relative to
    ↪ current working dir"
  default     = "./operator_values.yaml"
}
```

5.4.8 管理多个 TiDB 集群

一个 tidb-cluster 模块的实例对应一个 TiDB 集群，你可以通过编辑 clusters.tf 添加新的 tidb-cluster 模块实例来新增 TiDB 集群，示例如下：

```
module example-cluster {
  source = "../modules/aws/tidb-cluster"

  # The target EKS, required
  eks = local.eks
  # The subnets of node pools of this TiDB cluster, required
  subnets = local.subnets
  # TiDB cluster name, required
  cluster_name  = "example-cluster"

  # Helm values file
  override_values = file("example-cluster.yaml")
  # TiDB cluster version
  cluster_version           = "v3.0.0"
  # SSH key of cluster nodes
  ssh_key_name              = module.key-pair.key_name
  # PD replica number
  pd_count                  = 3
  # TiKV instance type
  pd_instance_type          = "t2.xlarge"
  # TiKV replica number
  tikv_count                = 3
  # TiKV instance type
  tikv_instance_type        = "t2.xlarge"
  # The storage class used by TiKV, if the TiKV instance type do not have
    ↪ local SSD, you should change it to storage class
  # TiDB replica number
  tidb_count                = 2
  # TiDB instance type
  tidb_instance_type        = "t2.xlarge"
  # Monitor instance type
```

```
  monitor_instance_type     = "t2.xlarge"
  # The version of tidb-cluster helm chart
  tidb_cluster_chart_version = "v1.0.0"
  # Decides whether or not to create the tidb-cluster helm release.
  # If this variable is set to false, you have to
  # install the helm release manually
  create_tidb_cluster_release = true
}
```

> **注意：**
>
> `cluster_name` 必须是唯一的。

你可以通过 `kubectl` 获取新集群的监控系统地址与 TiDB 地址。假如你希望让 Terraform 脚本输出这些地址，可以通过在 `outputs.tf` 中增加相关的输出项实现：

```
output "example-cluster_tidb-hostname" {
  value = module.example-cluster.tidb_hostname
}

output "example-cluster_monitor-hostname" {
  value = module.example-cluster.monitor_hostname
}
```

修改完成后，执行 `terraform init` 和 `terraform apply` 创建集群。

最后，只要移除 `tidb-cluster` 模块调用，对应的 TiDB 集群就会被销毁，EC2 资源也会随之释放。

5.4.9 仅管理基础设施

通过调整配置，你可以控制 Terraform 脚本只创建 Kubernetes 集群和 TiDB Operator。操作步骤如下：

- 修改 `clusters.tf` 中 TiDB 集群的 `create_tidb_cluster_release` 配置项：

```
hcl  module "default-cluster" { ...    create_tidb_cluster_release =
↪ false }
```

如上所示，当 `create_tidb_cluster_release` 设置为 `false` 时，Terraform 脚本不会创建和修改 TiDB 集群，但仍会创建 TiDB 集群所需的计算和存储资源。此时，你可以使用 Helm 等工具来独立管理集群。

注意：

在已经部署的集群上将 `create_tidb_cluster_release` 调整为 `false` 会导致已安装的 TiDB 集群被删除，对应的 TiDB 集群对象也会随之被删除。

5.4.10 销毁集群

可以通过如下命令销毁集群：

```
terraform destroy
```

注意：

- 该操作会销毁 EKS 集群以及部署在该 EKS 集群上的所有 TiDB 集群。
- 如果你不再需要存储卷中的数据，在执行 `terraform destroy` 后，你需要在 AWS 控制台手动删除 EBS 卷。

5.4.11 管理多个 Kubernetes 集群

本节详细介绍了如何管理多个 Kubernetes 集群（EKS），并在每个集群上部署一个或更多 TiDB 集群。

上述文档中介绍的 Terraform 脚本组合了多个 Terraform 模块：

- `tidb-operator` 模块，用于创建 EKS 集群并在 EKS 集群上安装配置TiDB Operator。
- `tidb-cluster` 模块，用于创建 TiDB 集群所需的资源池并部署 TiDB 集群。
- EKS 上的 TiDB 集群专用的 `vpc` 模块、`key-pair`模块和`bastion` 模块

管理多个 Kubernetes 集群的最佳实践是为每个 Kubernetes 集群创建一个单独的目录，并在新目录中自行组合上述 Terraform 模块。这种方式能够保证多个集群间的 Terraform 状态不会互相影响，也便于自由定制和扩展。下面是一个例子：

```
mkdir -p deploy/aws-staging
vim deploy/aws-staging/main.tf
```

`deploy/aws-staging/main.tf` 的内容可以是：

```
provider "aws" {
  region = "us-west-1"
}

## 创建一个 ssh key，用于登录堡垒机和 Kubernetes 节点
module "key-pair" {
  source = "../modules/aws/key-pair"

  name = "another-eks-cluster"
  path = "${path.cwd}/credentials/"
}

## 创建一个新的 VPC
module "vpc" {
  source = "../modules/aws/vpc"

  vpc_name = "another-eks-cluster"
}

## 在上面的 VPC 中创建一个 EKS 并部署 tidb-operator
module "tidb-operator" {
  source = "../modules/aws/tidb-operator"

  eks_name         = "another-eks-cluster"
  config_output_path = "credentials/"
  subnets          = module.vpc.private_subnets
  vpc_id           = module.vpc.vpc_id
  ssh_key_name     = module.key-pair.key_name
}

## 特殊处理，确保 helm 操作在 EKS 创建完毕后进行
resource "local_file" "kubeconfig" {
  depends_on      = [module.tidb-operator.eks]
  sensitive_content = module.tidb-operator.eks.kubeconfig
  filename        = module.tidb-operator.eks.kubeconfig_filename
}
provider "helm" {
  alias   = "eks"
  insecure = true
  install_tiller = false
  kubernetes {
    config_path = local_file.kubeconfig.filename
  }
}
```

```
## 在上面的 EKS 集群上创建一个 TiDB 集群
module "tidb-cluster-a" {
  source = "../modules/aws/tidb-cluster"
  providers = {
    helm = "helm.eks"
  }

  cluster_name = "tidb-cluster-a"
  eks          = module.tidb-operator.eks
  ssh_key_name = module.key-pair.key_name
  subnets      = module.vpc.private_subnets
}

## 在上面的 EKS 集群上创建另一个 TiDB 集群
module "tidb-cluster-b" {
  source = "../modules/aws/tidb-cluster"
  providers = {
    helm = "helm.eks"
  }

  cluster_name = "tidb-cluster-b"
  eks          = module.tidb-operator.eks
  ssh_key_name = module.key-pair.key_name
  subnets      = module.vpc.private_subnets
}

## 创建一台堡垒机
module "bastion" {
  source = "../modules/aws/bastion"

  bastion_name          = "another-eks-cluster-bastion"
  key_name              = module.key-pair.key_name
  public_subnets        = module.vpc.public_subnets
  vpc_id                = module.vpc.vpc_id
  target_security_group_id = module.tidb-operator.eks.
     ↪ worker_security_group_id
  enable_ssh_to_workers = true
}

## 输出 tidb-cluster-a 的 TiDB 服务地址
output "cluster-a_tidb-dns" {
  description = "tidb service endpoints"
  value       = module.tidb-cluster-a.tidb_hostname
}
```

```
## 输出 tidb-cluster-b 的监控地址
output "cluster-b_monitor-dns" {
  description = "tidb service endpoint"
  value       = module.tidb-cluster-b.monitor_hostname
}

## 输出堡垒机 IP
output "bastion_ip" {
  description = "Bastion IP address"
  value       = module.bastion.bastion_ip
}
```

上面的例子很容易进行定制，比如，假如你不需要堡垒机，便可以删去对 bastion 模块的调用。同时，项目中提供的 Terraform 模块均设置了合理的默认值，因此在调用这些 Terraform 模块时，你可以略去大部分的参数。

你可以参考默认的 Terraform 脚本来定制每个模块的参数，也可以参考每个模块的 variables.tf 文件来了解所有可配置的参数。

另外，这些 Terraform 模块可以很容易地集成到你自己的 Terraform 工作流中。假如你对 Terraform 非常熟悉，这也是我们推荐的一种使用方式。

注意：

- 由于 Terraform 本身的限制 (hashicorp/terraform#2430)，在你自己的 Terraform 脚本中，也需要保留上述例子中对 helm provider 的特殊处理。
- 创建新目录时，需要注意与 Terraform 模块之间的相对路径，这会影响调用模块时的 source 参数。
- 假如你想在 tidb-operator 项目之外使用这些模块，你需要确保 modules 目录中的所有模块的相对路径保持不变。

假如你不想自己写 Terraform 代码，也可以直接拷贝 deploy/aws 目录来创建新的 Kubernetes 集群。但要注意不能拷贝已经运行过 terraform apply 的目录 (已经有 Terraform 的本地状态)。这种情况下，推荐在拷贝前克隆一个新的仓库。

5.5 在 GCP GKE 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 GCP GKE 上部署 TiDB 集群。

> **警告：**
>
> 当前多磁盘聚合功能存在一些已知问题，不建议在生产环境中每节点配置一块以上磁盘。我们正在修复此问题。

5.5.1 环境准备

部署前，确认已安装以下软件：

- Git
- Google Cloud SDK
- Terraform >= 0.12
- kubectl >= 1.12
- Helm >= 2.11.0 且 < 2.16.4
- jq

5.5.2 配置

为保证部署顺利，需要提前进行一些配置。在开始配置 Google Cloud SDK、API、Terraform 前，先下载以下资源：

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

5.5.2.1 配置 Google Cloud SDK

安装 Google Cloud SDK 后，需要执行 `gcloud init` 进行初始化。

5.5.2.2 配置 API

如果使用的 GCP 项目是新项目，需确保以下 API 已启用：

```
gcloud services enable cloudresourcemanager.googleapis.com \
cloudbilling.googleapis.com iam.googleapis.com \
compute.googleapis.com container.googleapis.com
```

5.5.2.3 配置 Terraform

要执行 Terraform 脚本，需要设置以下 3 个环境变量。你可以等 Terraform 提示再输入，也可以提前在 `.tfvars` 文件中定义变量。

- `GCP_CREDENTIALS_PATH`：GCP 证书文件路径。

  - 建议另建一个服务账号给 Terraform 使用，参考创建与管理服务账号文档。`./create-service-account.sh` 会创建最低权限的服务账号。
  - 参考服务账号密钥文档来创建服务账号密钥。下面脚本中的步骤详细说明了如何使用 `deploy/gcp` 目录中提供的脚本执行此操作。或者，如果自己创建服务账号和密钥，可以在创建时选择 `JSON` 类型的密钥。下载的包含私钥的 `JSON` 文件即所需的证书文件。

- `GCP_REGION`：创建资源所在的区域，例如：`us-west1`。
- `GCP_PROJECT`：GCP 项目的名称。

要使用以上 3 个环境变量来配置 Terraform，可执行以下步骤：

1. 将 `GCP_REGION` 替换为你的 GCP region。

```
echo GCP_REGION=\"us-west1\" >> terraform.tfvars
```

2. 将 `GCP_PROJECT` 替换为你的 GCP 项目名称，确保连接的是正确的 GCP 项目。

```
echo "GCP_PROJECT=\"$(gcloud config get-value project)\"" >> terraform.
    ↪ tfvars
```

3. 初始化 Terraform。

```
terraform init
```

4. 为 Terraform 创建一个有限权限的服务账号，并设置证书路径。

```
./create-service-account.sh
```

Terraform 自动加载和填充匹配 `terraform.tfvars` 或 `*.auto.tfvars` 文件的变量。相关详细信息，请参阅 Terraform 文档。上述步骤会使用 `GCP_REGION` 和 `GCP_PROJECT` 填充 `terraform.tfvars` 文件，使用 `GCP_CREDENTIALS_PATH` 填充 `credentials.auto.tfvars` 文件。

5.5.3 部署 TiDB 集群

本小节介绍如何部署 TiDB 集群。

1. 确定实例类型。

   - 如果只是想试一下 TiDB，又不想花费太高成本，可以采用轻量级的配置：

```
cat small.tfvars >> terraform.tfvars
```

- 如果要对生产环境的部署进行 benchmark 测试，则建议采用生产级的配置：

```
cat prod.tfvars >> terraform.tfvars
```

prod.tfvars 会默认创建一个新的 VPC，两个子网和一个 f1-micro 实例作为堡垒机，以及使用以下实例类型作为工作节点的 GKE 集群：

  - 3 台 n1-standard-4 实例：部署 PD
  - 3 台 n1-highmem-8 实例：部署 TiKV
  - 3 台 n1-standard-16 实例：部署 TiDB
  - 3 台 n1-standard-2 实例：部署监控组件

如上所述，生产环境的部署需要 91 个 CPU，超过了 GCP 项目的默认配额。可以参考配额来增加项目配额。扩容同样需要更多 CPU。

> 注意：
> 工作节点的数量取决于指定 region 中可用区的数量。大部分 region 有 3 个可用区，但是 us-central1 有 4 个可用区。参考 Regions and zones 查看更多信息。参考自定义部分来自定义区域集群的节点池。

2. 启动脚本来部署 TiDB 集群：

```
terraform apply
```

> 注意：
> 如果未提前设置上文所述的 3 个环境变量，执行 terraform apply 过程中会有提示出现，要求对 3 个变量进行设置。详情请参考配置 Terraform。

整个过程可能至少需要 10 分钟。terraform apply 执行成功后，会输出类似如下的信息:

```
Apply complete! Resources: 23 added, 0 changed, 0 destroyed.

Outputs:

how_to_connect_to_default_cluster_tidb_from_bastion = mysql -h
    ↪ 172.31.252.20 -P 4000 -u root
how_to_ssh_to_bastion = gcloud compute ssh tidb-cluster-bastion --zone
    ↪ us-west1-b
how_to_set_reclaim_policy_of_pv_for_default_tidb_cluster_to_delete =
    ↪ kubectl --kubeconfig /.../credentials/kubeconfig_tidb-cluster get
    ↪  pvc -n tidb-cluster -o jsonpath='{.items[*].spec.volumeName}'|
    ↪ fmt -1 | xargs -I {} kubectl --kubeconfig /.../credentials/
    ↪ kubeconfig_tidb-cluster patch pv {} -p '{"spec":{"
    ↪ persistentVolumeReclaimPolicy":"Delete"}}'
```

```
kubeconfig_file = ./credentials/kubeconfig_tidb-cluster
monitor_lb_ip = 35.227.134.146
monitor_port = 3000
region = us-west1
tidb_version = v3.0.1
```

5.5.4 访问 TiDB 数据库

terraform apply 运行完成后，可执行以下步骤来访问 TiDB 数据库。注意用部署 TiDB 集群小节的输出信息替换 <> 部分的内容。

1. 通过 ssh 远程连接到堡垒机。

```
gcloud compute ssh <gke-cluster-name>-bastion --zone <zone>
```

2. 通过 MySQL 客户端来访问 TiDB 集群。

```
mysql -h <tidb_ilb_ip> -P 4000 -u root
```

> **注意：**
> 通过 MySQL 连接 TiDB 前，需要先安装 MySQL 客户端。

5.5.5 与 GKE 集群交互

你可以通过 kubectl 和 helm 使用 kubeconfig 文件 credentials/kubeconfig_<gke_cluster_name> 和 GKE 集群交互。交互方式主要有以下两种。

> **注意：**
> gke_cluster_name 默认为 tidb-cluster，可以通过 variables.tf 中 gke_name 修改。

- 指定 --kubeconfig 参数：

```
kubectl --kubeconfig credentials/kubeconfig_<gke_cluster_name> get po -n <tidb_cluster_name>
```

> **注意：**
> 下面这条命令使用的 `--kubeconfig` 参数至少需要 Helm 2.10.0 版本以上。

```
helm --kubeconfig credentials/kubeconfig_<gke_cluster_name> ls
```

- 设置 `KUBECONFIG` 环境变量：

```
export KUBECONFIG=$PWD/credentials/kubeconfig_<gke_cluster_name>
```

```
kubectl get po -n <tidb_cluster_name>
```

```
helm ls
```

5.5.6 升级 TiDB 集群

要升级 TiDB 集群，可执行以下步骤：

1. 编辑 `variables.tf` 文件，将 `tidb_version` 变量的值修改为更高版本。
2. 运行 `terraform apply`。

例如，要将 TiDB 集群升级到 3.0.0-rc.2，可修改 `tidb_version` 为 `v3.0.0-rc.2`：

```
variable "tidb_version" {
  description = "TiDB version"
  default     = "v3.0.0-rc.2"
}
```

升级过程会持续一段时间。你可以通过以下命令来持续观察升级进度：

```
kubectl --kubeconfig credentials/kubeconfig_<gke_cluster_name> get po -n <
    ↪ tidb_cluster_name> --watch
```

然后，你可以访问数据库并通过 `tidb_version()` 确认 TiDB 集群是否升级成功：

```
select tidb_version();
```

```
*************************** 1. row ***************************
tidb_version(): Release Version: v3.0.0-rc.2
Git Commit Hash: 06f3f63d5a87e7f0436c0618cf524fea7172eb93
Git Branch: HEAD
UTC Build Time: 2019-05-28 12:48:52
GoVersion: go version go1.12 linux/amd64
```

```
Race Enabled: false
TiKV Min Version: 2.1.0-alpha.1-ff3dd160846b7d1aed9079c389fc188f7f5ea13e
Check Table Before Drop: false
1 row in set (0.001 sec)
```

5.5.7 管理多个 TiDB 集群

一个 tidb-cluster 模块的实例对应一个 GKE 集群中的 TiDB 集群。要添加一个新的 TiDB 集群，可执行以下步骤：

1. 编辑 tidbclusters.tf 文件来添加一个 tidb-cluster 模块。

   例如：

```
module "example-tidb-cluster" {
providers = {
    helm = "helm.gke"
}
source                  = "../modules/gcp/tidb-cluster"
cluster_id              = module.tidb-operator.cluster_id
tidb_operator_id        = module.tidb-operator.tidb_operator_id
gcp_project             = var.GCP_PROJECT
gke_cluster_location    = local.location
gke_cluster_name        = <gke-cluster-name>
cluster_name            = <example-tidb-cluster>
cluster_version         = "v3.0.1"
kubeconfig_path         = local.kubeconfig
tidb_cluster_chart_version = "v1.0.0"
pd_instance_type        = "n1-standard-1"
tikv_instance_type      = "n1-standard-4"
tidb_instance_type      = "n1-standard-2"
monitor_instance_type   = "n1-standard-1"
pd_node_count           = 1
tikv_node_count         = 2
tidb_node_count         = 1
monitor_node_count      = 1
}
```

注意：

- 每个集群的 cluster_name 必须是唯一的。
- 为任一组件实际创建的总节点数等于配置文件中的节点数乘以该 region 中可用区的个数。

你可以通过 kubectl 获取创建的 TiDB 集群和监控组件的地址。如果你希望 Terraform 脚本打印此信息，可在 outputs.tf 中添加一个 output 配置项，如下所示：

```
output "how_to_connect_to_example_tidb_cluster_from_bastion" {
value = module.example-tidb-cluster.how_to_connect_to_tidb_from_bastion
}
```

上述配置可使该脚本打印出用于连接 TiDB 集群的命令。

2. 修改完成后，执行以下命令来创建集群。

```
terraform init
```

```
terraform apply
```

5.5.8 扩容

如果需要扩容 TiDB 集群，可执行以下步骤：

1. 按需修改 variables.tf 文件中的 tikv_count、tidb_count 变量。
2. 运行 terraform apply。

> **警告：**
>
> 由于缩容过程中无法确定哪个节点会被删除，因此目前不支持集群缩容。通过修改 tikv_count 来进行缩容可能会导致数据丢失。

扩容过程会持续几分钟，你可以通过以下命令来持续观察进度：

```
kubectl --kubeconfig credentials/kubeconfig_<gke_cluster_name> get po -n <
    ↪ tidb_cluster_name> --watch
```

例如，可以将 tidb_count 从 1 改为 2 来扩容 TiDB：

```
variable "tidb_count" {
  description = "Number of TiDB nodes per availability zone"
  default     = 2
}
```

注意：

增加节点数量会在每个可用区都增加节点。

5.5.9 自定义

你可以更改 variables.tf 中的默认值，例如集群名称和镜像版本等，但更建议在 terraform.tfvars 文件或其它相关文件中来指定值。

5.5.9.1 自定义 GCP 资源

GCP 允许 n1-standard-1 或者更大的实例类型挂载本地 SSD，这提供了更好的自定义特性。

5.5.9.2 自定义 TiDB 参数配置

Terraform 脚本为 GKE 中的 TiDB 集群提供了默认设置。你也可以在 tidbclusters.tf 中为每个 TiDB 集群指定一个覆盖配置 override_values 或者覆盖配置文件 override_values_file。如果同时配置两个变量，override_values 配置将生效，该自定义配置会覆盖默认设置，示例如下：

```
override_values = <<EOF
discovery:
  image: pingcap/tidb-operator:v1.0.1
  imagePullPolicy: IfNotPresent
  resources:
    limits:
      cpu: 250m
      memory: 150Mi
    requests:
      cpu: 30m
      memory: 30Mi
EOF
```

```
override_values_file = "./test-cluster.yaml"
```

集群默认使用 deploy/modules/gcp/tidb-cluster 模块中的 values/default.yaml 作为覆盖配置文件。

在 GKE 中，某些值不支持在 values.yaml 中自定义，包括集群版本、副本数、NodeSelector 以及 Tolerations。NodeSelector 和 Tolerations 由 Terraform 直接管理，以确保基础设施与 TiDB 集群之间的一致性。

如果需要自定义集群版本和副本数，可以修改 tidbclusters.tf 文件中每个 tidb-cluster module 的参数。

> **注意：**
>
> 自定义配置中，不建议在 values.yaml 中包含以下配置（tidb-cluster module 默认固定配置）：

```
pd:
  storageClassName: pd-ssd
tikv:
  stroageClassName: local-storage
tidb:
  service:
    type: LoadBalancer
    annotations:
      cloud.google.com/load-balancer-type: "Internal"
  separateSlowLog: true
monitor:
  storageClassName: pd-ssd
  persistent: true
  grafana:
    config:
      GF_AUTH_ANONYMOUS_ENABLED: "true"
    service:
      type: LoadBalancer
```

5.5.9.3 自定义 TiDB Operator

如果要自定义 TiDB Operator，可以使用 operator_helm_values 变量来指定覆盖配置或者使用 operator_helm_values_file 变量来指定覆盖配置文件。如果同时配置两个变量，operator_helm_values 配置将生效，该自定义配置会传递给 tidb-operator 模块，示例如下：

```
operator_helm_values = <<EOF
controllerManager:
  resources:
    limits:
      cpu: 250m
      memory: 150Mi
    requests:
      cpu: 30m
```

```
    memory: 30Mi
EOF
```

```
operator_helm_values_file = "./test-operator.yaml"
```

5.5.9.4 自定义日志

GKE 使用 Fluentd 作为其默认的日志收集工具，然后将日志转发到 Stackdriver。Fluentd 进程可能会占用大量资源，消耗大量的 CPU 和 RAM。Fluent Bit 是一种性能更高，资源占用更少的替代方案。与 Fluentd 相比，更建议在生产环境中使用 Fluent Bit。可参考在 GKE 集群中设置 Fluent Bit 的示例。

5.5.9.5 自定义节点池

集群是按区域 (regional) 而非按可用区 (zonal) 来创建的。也就是说，GKE 向每个可用区复制相同的节点池，以实现更高的可用性。但对于 Grafana 这样的监控服务来说，通常没有必要维护相同的可用性。你可以通过 gcloud 手动删除节点。

> **注意：**
>
> GKE 节点池通过实例组管理。如果你使用 `gcloud compute instances delete` 命令删除某个节点，GKE 会自动重新创建节点并将其添加到集群。

如果你需要从监控节点池中删掉一个节点，可采用如下步骤：

1. 获取托管的实例组和所在可用区。

```
gcloud compute instance-groups managed list | grep monitor
```

输出结果类似：

```
gke-tidb-monitor-pool-08578e18-grp us-west1-b zone gke-tidb-monitor-
    ↪ pool-08578e18 0 0        gke-tidb-monitor-pool-08578e18 no
gke-tidb-monitor-pool-7e31100f-grp us-west1-c zone gke-tidb-monitor-
    ↪ pool-7e31100f 1 1        gke-tidb-monitor-pool-7e31100f no
gke-tidb-monitor-pool-78a961e5-grp us-west1-a zone gke-tidb-monitor-
    ↪ pool-78a961e5 1 1        gke-tidb-monitor-pool-78a961e5 no
```

第一列是托管的实例组，第二列是所在的可用区。

2. 获取实例组中的实例名字。

```
gcloud compute instance-groups managed list-instances <the-name-of-the-
    ↪ managed-instance-group> --zone <zone>
```

示例：

```
gcloud compute instance-groups managed list-instances gke-tidb-monitor-
    ↪ pool-08578e18-grp --zone us-west1-b
```

输出结果类似：

```
NAME                                     ZONE       STATUS  ACTION
    ↪ INSTANCE_TEMPLATE                  VERSION_NAME LAST_ERROR
gke-tidb-monitor-pool-08578e18-c7vd us-west1-b RUNNING NONE gke-tidb-
    ↪ monitor-pool-08578e18
```

3. 通过指定托管的实例组和实例的名称来删掉该实例。

   例如：

```
gcloud compute instance-groups managed delete-instances gke-tidb-
    ↪ monitor-pool-08578e18-grp --instances=gke-tidb-monitor-pool-08578
    ↪ e18-c7vd --zone us-west1-b
```

5.5.10 销毁 TiDB 集群

如果你不想再继续使用 TiDB 集群，可以通过如下命令进行销毁：

```
terraform destroy
```

> 注意：
>
> 在执行 `terraform destroy` 过程中，可能会发生错误：`Error reading
> ↪ Container Cluster "tidb": Cluster "tidb" has status "
> ↪ RECONCILING" with message""`。当 GCP 升级 Kubernetes master 节点时会出现该问题。一旦问题出现，就无法删除集群，需要等待 GCP 升级结束，再次执行 `terraform destroy`。

5.5.10.1 删除磁盘

如果你不再需要之前的数据，并且想要删除正在使用的磁盘，有以下两种方法可以完成此操作：

- 手动删除：在 Google Cloud Console 中删除磁盘，或使用 gcloud 命令行工具执行删除操作。
- 自动删除：在执行 terraform destroy 之前将 Kubernetes 的 PV (Persistent Volume) 回收策略设置为 Delete，具体操作为在 terraform destroy 之前运行以下 kubectl 命令：

```
kubectl --kubeconfig /path/to/kubeconfig/file get pvc -n namespace-of-
    ↪ tidb-cluster -o jsonpath='{.items[*].spec.volumeName}'|fmt -1 |
    ↪ xargs -I {} kubectl --kubeconfig /path/to/kubeconfig/file patch
    ↪ pv {} -p '{"spec":{"persistentVolumeReclaimPolicy":"Delete"}}'
```

上述命令将获取 TiDB 集群命名空间中的 PVC (Persistent Volume Claim)，并将绑定的 PV 的回收策略设置为 Delete。在执行 terraform destroy 过程中删除 PVC 时，也会将磁盘删除。

下面是一个名为 change-pv-reclaimpolicy.sh 的脚本。相对于仓库根目录来说，它在 deploy/gcp 目录，简化了上述过程。

```
./change-pv-reclaimpolicy.sh /path/to/kubeconfig/file <tidb-cluster-
    ↪ namespace>
```

5.5.11 管理多个 Kubernetes 集群

本节介绍管理多个 Kubernetes 集群的最佳实践，其中每个 Kubernetes 集群都可以部署一个或多个 TiDB 集群。

在 TiDB 的案例中，Terraform 模块通常结合了几个子模块：

- tidb-operator：为 TiDB 集群提供 Kubernetes Control Plane 并部署 TiDB Operator。
- tidb-cluster：在目标 Kubernetes 集群中创建资源池并部署 TiDB 集群。
- 一个 vpc 模块，一个 bastion 模块和一个 project-credentials 模块：专门用于 GKE 上的 TiDB 集群。

管理多个 Kubernetes 集群的最佳实践有以下两点：

1. 为每个 Kubernetes 集群创建一个新目录。
2. 根据具体需求，使用 Terraform 脚本将上述模块进行组合。

如果采用了最佳实践，集群中的 Terraform 状态不会相互干扰，并且可以很方便地管理多个 Kubernetes 集群。示例如下 (假设已在项目根目录)：

```
mkdir -p deploy/gcp-staging && \
vim deploy/gcp-staging/main.tf
```

deploy/gcp-staging/main.tf 中的内容类似：

```
provider "google" {
  credentials = file(var.GCP_CREDENTIALS_PATH)
  region     = var.GCP_REGION
  project    = var.GCP_PROJECT
}

// required for taints on node pools
provider "google-beta" {
  credentials = file(var.GCP_CREDENTIALS_PATH)
  region     = var.GCP_REGION
  project    = var.GCP_PROJECT
}

locals {
  gke_name       = "another-gke-name"
  credential_path = "${path.cwd}/credentials"
  kubeconfig     = "${local.credential_path}/kubeconfig_${var.gke_name}"
}


module "project-credentials" {
  source = "../modules/gcp/project-credentials"

  path = local.credential_path
}

module "vpc" {
  source            = "../modules/gcp/vpc"
  create_vpc        = true
  gcp_project       = var.GCP_PROJECT
  gcp_region        = var.GCP_REGION
  vpc_name          = "${locals.gke_name}-vpc-network"
  private_subnet_name = "${locals.gke_name}-private-subnet"
  public_subnet_name = "${locals.gke_name}-public-subnet"
}

module "tidb-operator" {
  source             = "../modules/gcp/tidb-operator"
  gke_name           = locals.gke_name
  vpc_name           = module.vpc.vpc_name
  subnetwork_name    = module.vpc.private_subnetwork_name
  gcp_project        = var.GCP_PROJECT
  gcp_region         = var.GCP_REGION
  kubeconfig_path    = local.kubeconfig
```

```
  tidb_operator_version = "v1.0.0"
}

module "bastion" {
  source           = "../modules/gcp/bastion"
  vpc_name         = module.vpc.vpc_name
  public_subnet_name = module.vpc.public_subnetwork_name
  gcp_project      = var.GCP_PROJECT
  bastion_name     = "${locals.gke_name}-tidb-bastion"
}

## HACK: 强制使 Helm 依赖 GKE 集群
data "local_file" "kubeconfig" {
  depends_on = [module.tidb-operator.cluster_id]
  filename  = module.tidb-operator.kubeconfig_path
}
resource "local_file" "kubeconfig" {
  depends_on = [module.tidb-operator.cluster_id]
  content   = data.local_file.kubeconfig.content
  filename  = module.tidb-operator.kubeconfig_path
}

provider "helm" {
  alias        = "gke"
  insecure     = true
  install_tiller = false
  kubernetes {
    config_path = local_file.kubeconfig.filename
  }
}
module "tidb-cluster-a" {
  providers = {
    helm = "helm.gke"
  }
  source                  = "../modules/gcp/tidb-cluster"
  gcp_project             = var.GCP_PROJECT
  gke_cluster_location    = var.GCP_REGION
  gke_cluster_name        = locals.gke_name
  cluster_name            = "tidb-cluster-a"
  cluster_version         = "v3.0.1"
  kubeconfig_path         = module.tidb-operator.kubeconfig_path
  tidb_cluster_chart_version = "v1.0.0"
  pd_instance_type        = "n1-standard-1"
  tikv_instance_type      = "n1-standard-4"
  tidb_instance_type      = "n1-standard-2"
```

```
  monitor_instance_type   = "n1-standard-1"
}

module "tidb-cluster-b" {
  providers = {
    helm = "helm.gke"
  }
  source                  = "../modules/gcp/tidb-cluster"
  gcp_project             = var.GCP_PROJECT
  gke_cluster_location    = var.GCP_REGION
  gke_cluster_name        = locals.gke_name
  cluster_name            = "tidb-cluster-b"
  cluster_version         = "v3.0.1"
  kubeconfig_path         = module.tidb-operator.kubeconfig_path
  tidb_cluster_chart_version = "v1.0.0"
  pd_instance_type        = "n1-standard-1"
  tikv_instance_type      = "n1-standard-4"
  tidb_instance_type      = "n1-standard-2"
  monitor_instance_type   = "n1-standard-1"
}

output "how_to_ssh_to_bastion" {
  value= module.bastion.how_to_ssh_to_bastion
}

output "connect_to_tidb_cluster_a_from_bastion" {
  value = module.tidb-cluster-a.
     ↪ how_to_connect_to_default_cluster_tidb_from_bastion
}

output "connect_to_tidb_cluster_b_from_bastion" {
  value = module.tidb-cluster-b.
     ↪ how_to_connect_to_default_cluster_tidb_from_bastion
}
```

如上述代码所示，你可以在每个模块调用中省略几个参数，因为有合理的默认值，并且可以轻松地自定义配置。例如，如果你不需要调用堡垒机模块，将其删除即可。

如果要自定义每个字段，可使用以下两种方法中的一种：

- 直接修改 `*.tf` 文件中 `module` 的参数配置。
- 参考每个模块的 `variables.tf` 文件，了解所有可修改的参数，并在 `terraform.tfvars` 中设置自定义值。

注意：

- 创建新目录时，请注意其与 Terraform 模块的相对路径，这会影响模块调用期间的 `source` 参数。
- 如果要在 tidb-operator 项目之外使用这些模块，务必确保复制整个 `modules` 目录并保持目录中每个模块的相对路径不变。
- 由于 Terraform 的限制（参见 hashicorp/terraform # 2430），上面示例中添加了 **HACK: 强制使 Helm 依赖 GKE 集群部分对 Helm provider 进行处理。**如果自己编写 tf 文件，需要包含这部分内容。

如果你不愿意编写 Terraform 代码，还可以复制 `deploy/gcp` 目录来创建新的 Kubernetes 集群。但需要注意，不能复制已被执行 `terraform apply` 命令的目录。在这种情况下，建议先克隆新的仓库再复制目录。

5.6 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。

5.6.1 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 >= 2.11.0 且 < 2.16.4
- 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。

5.6.1.1 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

5.6.2 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
  - 一块 100 GB 的云盘用作监控数据存储

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。

5.6.3 安装部署

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=<YOUR_REGION> && \
export TF_VAR_ALICLOUD_ACCESS_KEY=<YOUR_ACCESS_KEY> && \
export TF_VAR_ALICLOUD_SECRET_KEY=<YOUR_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
```

```
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 = 121.199.195.236:3000
region = cn-hangzhou
ssh_key_file = /tidb-operator/deploy/aliyun/credentials/my-cluster-keyZ
    ↪ .pem
tidb_endpoint = 172.21.5.171:4000
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
```

5.6.4 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_slb_ip> -P 4000 -u root
```

5.6.5 监控

访问 <monitor_endpoint> 就可以查看相关的 Grafana 监控面板。相关信息可在安装完成后的输出中找到。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 deploy/modules/aliyun/tidb-cluster/values/default.yaml 文件里 monitor.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。

5.6.6 升级 TiDB 集群

设置 variables.tf 中的 tidb_version 参数，并再次运行 terraform apply 即可完成升级。

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 <tidb_cluster_name> -o wide --watch
```

5.6.7 TiDB 集群水平伸缩

按需修改 variables.tf 中的 tikv_count 和 tidb_count 数值，再次运行 terraform apply 即可完成 TiDB 集群的水平伸缩。

5.6.8 销毁集群

```
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：
> 监控组件挂载的云盘需要在阿里云管理控制台中手动删除。

5.6.9 配置

5.6.9.1 配置 TiDB Operator

通过调整 `variables.tf` 内的值来配置 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。

5.6.9.2 配置 TiDB 集群

TiDB 集群会使用 `./my-cluster.yaml` 作为集群的 `values.yaml` 配置文件，修改该文件即可配置 TiDB 集群。支持的配置项可参考Kubernetes 上的 TiDB 集群配置。

5.6.10 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
  override_values         = file("dev-cluster.yaml")
}

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
  override_values         = file("staging-cluster.yaml")
}
```

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
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

5.6.11 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    = <YOUR_REGION>
    access_key = <YOUR_ACCESS_KEY>
    secret_key = <YOUR_SECRET_KEY>
}

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

    region         = <YOUR_REGION>
    access_key     = <YOUR_ACCESS_KEY>
    secret_key     = <YOUR_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。

5.6.12 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。

5.7 访问 Kubernetes 上的 TiDB 集群

在 Kubernetes 集群内访问 TiDB 时，使用 TiDB service 域名 `<release-name>-tidb.<namespace>` 即可。若需要在集群外访问，则需将 TiDB 服务端口暴露出去。在 `tidb-cluster` Helm chart 中，通过 `values.yaml` 文件中的 `tidb.service` 字段进行配置：

```
tidb:
  service:
    type: NodePort
    # externalTrafficPolicy: Cluster
    # annotations:
    # cloud.google.com/load-balancer-type: Internal
```

5.7.1 NodePort

在没有 LoadBalancer 时，可选择通过 NodePort 暴露。NodePort 有两种模式：

- `externalTrafficPolicy=Cluster`：集群所有的机器都会给 TiDB 分配 TCP 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 加同一个端口访问 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 TCP 端口，用于访问本地的 TiDB 实例

  使用 `Local` 模式时，建议打开 tidb-scheduler 的 `StableScheduling` 特性。tidb-scheduler 会尽可能在升级过程中将新 TiDB 实例调度到原机器，这样集群外的客户端便不需要在 TiDB 重启后更新配置。

5.7.1.1 查看 NodePort 模式下对外暴露的 IP/PORT

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 <release-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=<release-name>" -ojsonpath="{range .
    ↪ items[*]}{.spec.nodeName}{'\n'}{end}"
```

5.7.2 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。

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

5.8 TiDB Binlog 运维

本文档介绍如何在 Kubernetes 上运维 TiDB 集群的 TiDB Binlog。

5.8.1 运维准备

- 部署 TiDB Operator；
- 安装 Helm 并配置 PingCAP 官方 chart 仓库。

5.8.2 启用 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：

1. 按照以下说明修改 `values.yaml` 文件：

    - 将 `binlog.pump.create` 的值设为 `true`。
    - 将 `binlog.drainer.create` 的值设为 `true`。
    - 将 `binlog.pump.storageClassName` 和 `binlog.drainer.storageClassName` 设为所在 Kubernetes 集群上可用的 `storageClass`。
    - 将 `binlog.drainer.destDBType` 设为所需的下游存储类型。
      TiDB Binlog 支持三种下游存储类型：
        - PersistenceVolume：默认的下游存储类型。可通过修改 `binlog.drainer.storage` 来为 `drainer` 配置大 PV。
        - 与 MySQL 兼容的数据库：通过将 `binlog.drainer.destDBType` 设置为 `mysql` 来启用。同时，必须在 `binlog.drainer.mysql` 中配置目标数据库的地址和凭据。
        - Apache Kafka：通过将 `binlog.drainer.destDBType` 设置为 `kafka` 来启用。同时，必须在 `binlog.drainer.kafka` 中配置目标集群的 zookeeper 地址和 Kafka 地址。

2. 为 TiDB 与 Pump 组件设置亲和性和反亲和性：

> 注意：
> 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 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。

注意：

<release-name> 需要替换为目标 tidb-cluster 的 Helm release name。

- **将 tidb.affinity 按照如下设置：**

```
tidb:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - 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:
                  - <release-name>
          topologyKey: kubernetes.io/hostname
```

- **将 binlog.pump.affinity 按照如下设置：**

```
binlog:
  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:
              - <release-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:
                - <release-name>
            topologyKey: kubernetes.io/hostname
```

3. 创建一个新的 TiDB 集群或更新现有的集群：

    - 创建一个启用 TiDB Binlog 的 TiDB 新集群：

```
helm install pingcap/tidb-cluster --name=<release-name> --namespace
    ↪ =<namespace> --version=<chart-version> -f <values-file>
```

    - 更新现有的 TiDB 集群以启用 TiDB Binlog：

> **注意：**
> 如果设置了 TiDB 组件的亲和性，那么更新现有的 TiDB 集群将引起 TiDB 集群中的 TiDB 组件滚动更新。

```
helm upgrade <release-name> pingcap/tidb-cluster --version=<chart-
    ↪ version> -f <values-file>
```

5.8.3 部署多个 drainer

默认情况下，仅创建一个下游 drainer。可安装 `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 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: v3.0.0
storageClassName: local-storage
storage: 10Gi
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 = "slave-tidb"
  user = "root"
  password = ""
  port = 4000
```

clusterName 和 clusterVersion 必须匹配所需的源 TiDB 集群。

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 pingcap/tidb-drainer --name=<release-name> --namespace=<
    ↪ namespace> --version=<chart-version> -f values.yaml
```

> 注意：
> 该 chart 必须与源 TiDB 集群安装在相同的命名空间中。

6 配置

6.1 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：
> 以下功能只在第一次创建集群时有作用，集群创建之后再设置或修改不会生效。

6.1.1 设置初始化账号和密码

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：

1. 创建 Namespace

   在部署集群前通过下面命令创建 Namespace：

```
kubectl create namespace <namespace>
```

2. 创建 Secret

   在部署集群前通过下面命令创建 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-passowrd> --
    ↪ 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 tidb.initSql 中设置。

3. 设置允许访问 TiDB 的主机

   在部署集群前可以通过 tidb.permitHost 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。

```
tidb:
  passwordSecretName: tidb-secret
  permitHost: <mysql-client-host-name>
```

4. 部署集群

   创建 Secret 之后，通过下面命令部署集群：

```
helm install pingcap/tidb-cluster -f values.yaml --name=<release-name>
    ↪ --namespace=<namespace> --version=<chart-version>
```

以上命令指定 tidb.passwordSecretName 之后，创建的集群会自动创建一个初始化的 Job，该 Job 在集群创建过程中会尝试利用提供的 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。注意由于 Job 创建时 TiDB 集群的 Pod 还没完全创建，所以可能会失败几次，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。

6.1.2 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 tidb.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：

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

将上述内容保存到 values.yaml 文件，然后执行下面命令部署集群：

```
helm install pingcap/tidb-cluster -f values.yaml --name=<release-name> --
    ↪ namespace=<namespace> --version=<chart-version>
```

> **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。

6.2 Kubernetes 上的 TiDB 集群配置

本文介绍 Kubernetes 上 TiDB 集群的配置参数、资源配置，以及容灾配置。

6.2.1 配置参数

TiDB Operator 使用 Helm 部署和管理 TiDB 集群。通过 Helm 获取的配置文件默认提供了基本的配置，通过这个基本配置，可以快速启动一个 TiDB 集群。但是如果用户需要特殊配置或是用于生产环境，则需要根据以下配置参数列表手动配置对应的配置项。

> **注意：**
>
> 下文用 `values.yaml` 指代要修改的 TiDB 集群配置文件。

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.	TiDB 集群	pingcap/
image	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	tidb-operator:v1.0.0-beta.3
discovery.	PD 服务发现组件镜像的拉取策略	IfNotPresent
discovery.	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：nil 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	info
	如果 TiDB Operator 版本 > v1.0.0-beta.3，请通过 pd.config 配置： [log] level = "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 nodeSelector	{}

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 node-Selector	{}

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. ↪ readpoolCoprocessorConcurrency	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 --from literal = root =< 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 nodeSelector	{}
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 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.mysqlNodePort	TiDB 服务暴露的 mysql NodePort 端口	
tidb.service.exposeStatus	TiDB 服务是否暴露状态端口	true
tidb.service.statusNodePort	指定 TiDB 服务的状态端口暴露的 NodePort	

参数名	说明	默认值
`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 side-car 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

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

6.2.2 资源配置说明

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。

6.2.3 容灾配置说明

TiDB 是分布式数据库，它的容灾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容灾的配置。

6.2.3.1 TiDB 服务的容灾

TiDB 服务容灾本质上基于 Kubernetes 的调度功能来实现的，为了优化调度，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 服务的容灾，而且目前 TiDB Cluster 使用该调度器作为默认调度器，设置项是上述列表中的 schedulerName 配置项。

其它层面的容灾（例如 rack，zone，region）是通过 Affinity 的 PodAntiAffinity 来保证，通过 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，从而达到容灾的目的，Affinity 的使用参考：Affinity & AntiAffinity。

下面是一个典型的容灾设置例子：

```
affinity:
 podAntiAffinity:
   preferredDuringSchedulingIgnoredDuringExecution:
   # this term works when the nodes have the label named region
   - weight: 10
     podAffinityTerm:
       labelSelector:
         matchLabels:
           app.kubernetes.io/instance: <release name>
           app.kubernetes.io/component: "pd"
       topologyKey: "region"
       namespaces:
       - <helm namespace>
   # this term works when the nodes have the label named zone
   - weight: 20
     podAffinityTerm:
       labelSelector:
         matchLabels:
           app.kubernetes.io/instance: <release name>
           app.kubernetes.io/component: "pd"
       topologyKey: "zone"
       namespaces:
       - <helm namespace>
   # this term works when the nodes have the label named rack
```

```
  - weight: 40
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app.kubernetes.io/instance: <release name>
          app.kubernetes.io/component: "pd"
      topologyKey: "rack"
      namespaces:
      - <helm namespace>
  # this term works when the nodes have the label named kubernetes.io/
      ↪ hostname
  - weight: 80
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app.kubernetes.io/instance: <release name>
          app.kubernetes.io/component: "pd"
      topologyKey: "kubernetes.io/hostname"
      namespaces:
      - <helm namespace>
```

6.2.3.2 数据的容灾

在开始数据容灾配置前，首先请阅读集群拓扑信息配置。该文档描述了 TiDB 集群数据容灾的实现原理。

在 Kubernetes 上支持数据容灾的功能，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 `hostname`，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 <nodeName> region=<regionName> zone=<zoneName> rack
    ↪ =<rackName> kubernetes.io/hostname=<hostName>
```

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。

6.3 Kubernetes 上的 TiDB 集群备份配置

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。

6.3.1 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）

6.3.2 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

6.3.3 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

6.3.4 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>
```

6.3.5 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。

6.3.6 storage.size

- 持久卷的空间大小
- 默认：“100Gi”

6.3.7 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

6.3.8 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

6.3.9 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。

6.3.10 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
```

6.3.11 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。

6.3.12 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

6.3.13 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>
```

6.4 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。

6.4.1 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`。

6.4.2 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-name> -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>
   ```

6.4.3 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。创建流程如下：

1. 参考 Kubernetes 提供的操作文档，在集群节点中预分配本地存储。

2. 部署 local-volume-provisioner 程序。

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

   通过下面命令查看 Pod 和 PV 状态：

   ```
   kubectl get po -n kube-system -l app=local-volume-provisioner && \
   kubectl get pv | grep local-storage
   ```

   local-volume-provisioner 会为发现目录 (discovery directory) 下的每一个挂载点创建一个 PV。注意，在 GKE 上，默认只能创建大小为 375 GiB 的本地卷。

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

6.4.3.1 最佳实践

- Local 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会比较恰当，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或者每个存储卷使用一块物理盘来实现

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

6.4.3.2 示例

如果监控、TiDB Binlog、备份等组件都使用本地盘存储数据，可以挂载普通 SAS 盘，并分别创建不同的 `StorageClass` 供这些组件使用，具体操作如下：

- 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/disks` 目录，后续创建 `local-storage` StorageClass。

> 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，后续创建 `backup-storage` StorageClass。

> 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，每次Ad-hoc 全量备份会使用 1 个 PV，所有定时全量备份会共用 1 个 PV。

- 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，后续创建 `shared-ssd-storage` StorageClass。

> 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 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，后续创建 ssd-storage StorageClass。

盘挂载完成后，需要根据上述磁盘挂载情况修改 local-volume-provisioner yaml 文件，配置发现目录并创建必要的 StorageClass。以下是根据上述挂载修改的 yaml 文件示例：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "local-storage"
provisioner: "kubernetes.io/no-provisioner"
volumeBindingMode: "WaitForFirstConsumer"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "ssd-storage"
provisioner: "kubernetes.io/no-provisioner"
volumeBindingMode: "WaitForFirstConsumer"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "shared-ssd-storage"
provisioner: "kubernetes.io/no-provisioner"
volumeBindingMode: "WaitForFirstConsumer"
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: "backup-storage"
provisioner: "kubernetes.io/no-provisioner"
volumeBindingMode: "WaitForFirstConsumer"
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-provisioner-config
  namespace: kube-system
data:
  nodeLabelsForPV: |
    - kubernetes.io/hostname
  storageClassMap: |
    shared-ssd-storage:
```

```
    hostDir: /mnt/sharedssd
    mountDir: /mnt/sharedssd
  ssd-storage:
    hostDir: /mnt/ssd
    mountDir: /mnt/ssd
  local-storage:
    hostDir: /mnt/disks
    mountDir: /mnt/disks
  backup-storage:
    hostDir: /mnt/backup
    mountDir: /mnt/backup
---

......

        volumeMounts:

          ......

          - mountPath: /mnt/ssd
            name: local-ssd
            mountPropagation: "HostToContainer"
          - mountPath: /mnt/sharedssd
            name: local-sharedssd
            mountPropagation: "HostToContainer"
          - mountPath: /mnt/disks
            name: local-disks
            mountPropagation: "HostToContainer"
          - mountPath: /mnt/backup
            name: local-backup
            mountPropagation: "HostToContainer"
    volumes:

      ......

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

```
        hostPath:
          path: /mnt/backup
......
```

最后通过 kubectl apply 命令部署 local-volume-provisioner 程序。

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

后续创建 TiDB 集群或备份等组件的时候，再配置相应的 StorageClass 供其使用。

6.4.4 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。

PV 保留策略是 Retain 时，如果确认某个 PV 的数据可以被删除，需要通过下面的操作来删除 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。

6.5 Kubernetes 上的 TiDB Binlog Drainer 配置

本文档介绍 Kubernetes 上 TiDB Binlog drainer 的配置参数。

6.5.1 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
clusterName	源 TiDB 集群的名称	demo
clusterVersion	源 TiDB 集群的版本	v3.0.1
baseImage	TiDB Binlog 的基础镜像	pingcap/tidb-binlog
imagePullPolicy	镜像的拉取策略	IfNotPresent

参数	说明	默认值
logLevel	drainer 进程的日志级别	info
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
disableDispatch	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	0
config	传递到 drainer 的配置文件。详情可参阅 drainer.toml	（见下文）

参数	说明	默认值
`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"
```

7 Kubernetes 上的 TiDB 集群监控

基于 Kubernetes 环境部署的 TiDB 集群监控可以大体分为两个部分：对 TiDB 集群本身的监控、对 Kubernetes 集群及 TiDB Operator 的监控。本文将对两者进行简要说明。

7.1 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，对于每个 TiDB 集群，会同时创建、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。

监控数据默认没有持久化，如果由于某些原因监控容器重启，已有的监控数据会丢失。可以在 `values.yaml` 中设置 `monitor.persistent` 为 `true` 来持久化监控数据。开启此选项时应将 `storageClass` 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，否则仍然会存在数据丢失的风险。

在 TiDB 集群监控中有一些监控系统配置的细节可供参考。

7.1.1 查看监控面板

可以通过 `kubectl port-forward` 查看监控面板：

```
kubectl port-forward -n <namespace> svc/<release-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`。

Grafana 服务默认通过 `NodePort` 暴露，如果 Kubernetes 集群支持负载均衡器，你可以在 `values.yaml` 中将 `monitor.grafana.service.type` 修改为 `LoadBalancer`，然后在执行 `helm upgrade` 后通过负载均衡器访问面板。

如果不需要使用 Grafana，可以在部署时在 `values.yaml` 中将 `monitor.grafana.create` 设置为 `false` 来节省资源。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。

7.1.2 访问监控数据

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/<release-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。

Prometheus 服务默认通过 NodePort 暴露，如果 Kubernetes 集群支持负载均衡器，你可以在 values.yaml 中将 monitor.prometheus.service.type 修改为 LoadBalancer，然后在执行 helm upgrade 后通过负载均衡器访问监控数据。

7.2 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。

7.2.1 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。

7.2.2 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。

7.3 报警配置

7.3.1 TiDB 集群报警

在随 TiDB 集群部署 Prometheus 时，会自动导入一些默认的报警规则，可以通过浏览器访问 Prometheus 的 Alerts 页面查看当前系统中的所有报警规则和状态。

我们目前暂不支持报警规则的自定义配置，如果确实需要修改报警规则，可以手动下载 charts 进行修改。

默认的 Prometheus 和报警配置不能发送报警消息，如需发送报警消息，可以使用任意支持 Prometheus 报警的工具与其集成。推荐通过 AlertManager 管理与发送报警消息。

如果在你的现有基础设施中已经有可用的 AlertManager 服务，可以在 `values.yaml` 文件中修改 `monitor.prometheus.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。

7.3.2 Kubernetes 报警

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。

8 运维

8.1 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。

要销毁 TiDB 集群，执行以下命令：

```
helm delete <release-name> --purge
```

上述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。

```
kubectl delete pvc -n <namespace> -l app.kubernetes.io/instance=<release-
    ↪ 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=<release-name> -o
    ↪  name | xargs -I {} kubectl patch {} -p '{"spec":{"
    ↪ persistentVolumeReclaimPolicy":"Delete"}}'
```

8.2 重启 Kubernetes 上的 TiDB 集群

本文描述了如何强制重启 Kubernetes 集群上的 TiDB 集群，包括重启某个 Pod，重启某个组件的所有 Pod 和重启 TiDB 集群的所有 Pod。

> **注意：**
>
> TiDB Operator v1.0.x 版本只支持强制重启 Pod。
>
> - 在强制重启 PD Pod 过程中，如果被重启的 PD Pod 是 Leader，重启过程不会自动迁移 Leader，这会导致 PD 服务短时间中断。
> - 在强制重启 TiKV Pod 过程中，不会自动迁移 TiKV 的 Region Leader，会导致访问对应数据的请求异常。
> - 在强制重启 TiDB Pod 过程中，会导致访问对应 TiDB 的请求失败。

8.2.1 强制重启某个 Pod

要强制重启某个 Pod，执行以下命令：

```
kubectl delete pod -n <namespace> <pod-name>
```

8.2.2 强制重启某个组件的所有 Pod

通过以下命令可以列出组件目前有哪些 Pod：

```
kubectl get pod -n <namespace> -l app.kubernetes.io/component=<component-
    ↪ name>
```

要强制重启某个组件的所有 Pod，执行以下命令：

```
kubectl delete pod -n <namespace> -l app.kubernetes.io/component=<component-
    ↪ name>
```

把 <component-name> 分别替换为 pd、tidb、tikv，可以分别强制重启 PD、TiDB、TiKV 组件所有 Pod。

8.2.3 强制重启 TiDB 集群的所有 Pod

通过以下命令可以列出 TiDB 集群目前有哪些 Pod，包括 monitor、discovery 等：

```
kubectl get pod -n <namespace> -l app.kubernetes.io/instance=<tidb-cluster-
    ↪ name>
```

要强制重启 TiDB 集群的所有 Pod，包括 monitor、discovery 等，执行以下命令：

```
kubectl delete pod -n <namespace> -l app.kubernetes.io/instance=<tidb-
    ↪ cluster-name>
```

8.3 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 实例的不同特性，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
- tkctl
- jq

> 注意：
>
> 长期维护节点前，需要保证 Kubernetes 集群的剩余资源足够运行 TiDB 集群。

8.3.1 维护 PD 和 TiDB 实例所在节点

PD 和 TiDB 实例的迁移较快，可以采取主动驱逐实例到其它节点上的策略进行节点维护：

1. 检查待维护节点上是否有 TiKV 实例：

```
kubectl get pod --all-namespaces -o wide | grep <node-name>
```

假如存在 TiKV 实例，请参考维护 TiKV 实例所在节点。

2. 使用 kubectl cordon 命令防止新的 Pod 调度到待维护节点上：

```
kubectl cordon <node-name>
```

3. 使用 kubectl drain 命令将待维护节点上的数据库实例迁移到其它节点上：

```
kubectl drain <node-name> --ignore-daemonsets --delete-local-data
```

运行后，该节点上的 TiDB 实例会自动迁移到其它可用节点上，PD 实例则会在 5 分钟后触发自动故障转移补齐节点。

4. 此时，假如希望下线该 Kubernetes 节点，则可以将该节点删除：

```
kubectl delete node <node-name>
```

假如希望恢复 Kubernetes 节点，则需要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。

5. 使用 kubectl uncordon 命令解除节点的调度限制：

```
kubectl uncordon <node-name>
```

6. 观察 Pod 是否全部恢复正常运行：

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

或者：

```
watch tkctl get all
```

Pod 恢复正常运行后，操作结束。

8.3.2 维护 TiKV 实例所在节点

TiKV 实例迁移较慢，并且会对集群造成一定的数据迁移负载，因此在维护 TiKV 实例所在节点前，需要根据维护需求选择操作策略：

- 假如是维护短期内可恢复的节点，则不需要迁移 TiKV 节点，在维护结束后原地恢复节点；
- 假如是维护短期内不可恢复的节点，则需要规划 TiKV 的迁移工作。

8.3.2.1 维护短期内可恢复的节点

针对短期维护，我们可以通过调整 PD 集群的 `max-store-down-time` 配置来增大集群所允许的 TiKV 实例下线时间，在此时间内维护完毕并恢复 Kubernetes 节点后，所有该节点上的 TiKV 实例会自动恢复。

```
kubectl port-forward svc/<CLUSTER_NAME>-pd 2379:2379
```

```
pd-ctl -d config set max-store-down-time 10m
```

调整 `max-store-down-time` 到合理的值后，后续的操作方式与维护 PD 和 TiDB 实例所在节点相同。

8.3.2.2 维护短期内不可恢复的节点

针对短期内不可恢复的节点维护，如节点长期下线等情形，需要使用 `pd-ctl` 主动通知 TiDB 集群下线对应的 TiKV 实例，再手动解除 TiKV 实例与该节点的绑定。

1. 使用 `kubectl cordon` 命令防止新的 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 实例：

   ```
   tkctl get -A tikv | grep <node-name>
   ```

3. 使用 `pd-ctl` 主动下线 TiKV 实例。

> **注意：**
> 下线 TiKV 实例前，需要保证集群中剩余的 TiKV 实例数不少于 PD 配置中的 TiKV 数据副本数（配置项：`max-replicas`）。假如不符合该条件，需要先操作扩容 TiKV。

查看 TiKV 实例的 `store-id`：

```
kubectl get tc <CLUSTER_NAME> -ojson | jq '.status.tikv.stores | .[] |
    ↪ select ( .podName == "<POD_NAME>" ) | .id'
```

下线实例：

```
kubectl port-forward svc/<CLUSTER_NAME>-pd 2379:2379
```

```
pd-ctl -d store delete <ID>
```

4. 等待 store 状态（state_name）转移为 Tombstone：

```
watch pd-ctl -d store <ID>
```

5. 解除 TiKV 实例与节点本地盘的绑定。

查询 Pod 使用的 PesistentVolumeClaim：

```
kubectl get -n <namespace> pod <pod_name> -ojson | jq '.spec.volumes |
    ↪ .[] | select (.name == "tikv") | .persistentVolumeClaim.claimName
    ↪ '
```

删除该 PesistentVolumeClaim：

```
kubectl delete -n <namespace> pvc <pvc_name>
```

6. 删除 TiKV 实例：

```
kubectl delete -n <namespace> pod <pod_name>
```

7. 观察该 TiKV 实例是否正常调度到其它节点上：

```
watch kubectl -n <namespace> get pod -o wide
```

假如待维护节点上还有其它 TiKV 实例，则重复同样的操作步骤直到所有的 TiKV 实例都迁移到其它节点上。

8. 确认节点不再有 TiKV 实例后，再逐出节点上的其它实例：

```
kubectl drain <node-name> --ignore-daemonsets --delete-local-data
```

9. 再次确认节点不再有任何 TiKV、TiDB 和 PD 实例运行：

```
kubectl get pod --all-namespaces | grep <node-name>
```

10. 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>
```

至此，操作完成。

8.4 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。

8.4.1 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，v4.0.0-rc。

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。

8.4.2 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。

8.4.2.1 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
```

8.4.2.2 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。

8.4.2.3 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 Pod 已经启动：

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

8.4.2.4 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。

8.4.2.5 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。

8.4.2.6 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}
```

8.4.2.7 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}
```

8.5 备份与恢复

8.5.1 基于 Helm Charts 实现的 TiDB 集群备份与恢复

本文详细描述了如何对 Kubernetes 上的 TiDB 集群进行数据备份和数据恢复。本文使用的备份恢复方式是基于 Helm Charts 实现的。

Kubernetes 上的 TiDB 集群支持两种备份策略：

- 全量备份（定时执行或 Ad-hoc）：使用 mydumper 获取集群的逻辑备份；
- 增量备份：使用 TiDB Binlog 将 TiDB 集群的数据实时复制到其它数据库中或实时获得增量数据备份；

目前，Kubernetes 上的 TiDB 集群只对 mydumper 获取的全量备份数据提供自动化的数据恢复操作。恢复 TiDB-Binlog 获取的增量数据需要手动进行。

8.5.1.1 全量备份

全量备份使用 mydumper 来获取 TiDB 集群的逻辑备份数据。全量备份任务会创建一个 PVC (PersistentVolumeClaim) 来存储数据。

默认配置下，备份任务使用 PV (Persistent Volume) 来存储备份数据。你也可以通过修改配置将备份数据存储到 Google Cloud Storage，Ceph Object Storage 或 Amazon S3 中，在这种情况下，数据在上传到对象存储前，会临时存储在 PV 中。参考Kubernetes 上的 TiDB 集群备份配置 了解所有的配置选项。

你可以配置一个定时执行的全量备份任务，也可以临时执行一个全量备份任务。

8.5.1.1.1 定时全量备份

定时全量备份是一个与 TiDB 集群一同创建的定时任务，它会像 crontab 任务那样周期性地运行。

你可以修改 TiDB 集群的 values.yaml 文件来配置定时全量备份：

1. 将 scheduledBackup.create 设置为 true；
2. 将 scheduledBackup.storageClassName 设置为用于存储数据的 PV 的 storageClass ↪ ；

> 注意：
> 你必须将定时全量备份使用的 PV 的 reclaim policy 设置为 Retain 来确保你的数据安全。

3. 按照 Cron 格式设置 scheduledBackup.schedule 来定义任务的执行周期与时间；

4. 创建一个包含数据库用户名和密码的 Kubernetes Secret 该用户必须拥有数据备份所需的数据库相关权限，同时，将 scheduledBackup.secretName 设置为该 Secret 的名字 (默认为 backup-secret):

```
kubectl create secret generic backup-secret -n <namespace> --from-
    ↪ literal=user=<user> --from-literal=password=<password>
```

5. 通过 helm install 创建一个配置了定时全量备份的 TiDB 集群，针对现有集群，则使用 helm upgrade 为集群打开定时全量备份:

```
helm upgrade <release_name> pingcap/tidb-cluster -f values.yaml --
    ↪ version=<tidb-operator-version>
```

8.5.1.1.2 Ad-hoc 全量备份

Ad-hoc 全量备份封装在 pingcap/tidb-backup 这个 Helm chart 中。根据 values.yaml 文件中的 mode 配置，该 chart 可以执行全量备份或数据恢复。我们会在数据恢复一节中描述如何执行数据恢复。

你可以通过下面的步骤执行一次 Ad-hoc 全量备份:

1. 修改 values.yaml:

    - 将 clusterName 设置为目标 TiDB 集群名字;
    - 将 mode 设置为 backup;
    - 将 storage.className 设置为用于存储数据的 PV 的 storageClass;
    - 根据数据量调整 storage.size;

> 注意:
> 你必须将 Ad-hoc 全量备份使用的 PV 的 reclaim policy 设置为 Retain 来确保你的数据安全。

2. 创建一个包含数据库用户名和密码的 Kubernetes Secret，该用户必须拥有数据备份所需的数据库相关权限，同时，将 values.yaml 中的 secretName 设置为该 Secret 的名字 (默认为 backup-secret):

```
kubectl create secret generic backup-secret -n <namespace> --from-
    ↪ literal=user=<user> --from-literal=password=<password>
```

3. 使用下面的命令执行一次 Ad-hoc 全量备份:

```
helm install pingcap/tidb-backup --name=<backup-name> --namespace=<
    ↪ namespace> -f values.yaml --version=<tidb-operator-version>
```

8.5.1.1.3 查看备份

对于存储在 PV 中的备份，你可以使用下面的命令进行查看：

```
kubectl get pvc -n <namespace> -l app.kubernetes.io/component=backup,pingcap
    ↪ .com/backup-cluster-name=<cluster-name>
```

假如你将数据存储在 Google Cloud Storage，Ceph Object Storage 或 Amazon S3 中，你可以用这些存储系统自带的图形界面或命令行工具查看全量备份。

8.5.1.2 数据恢复

使用 pingcap/tidb-backup 这个 Helm chart 进行数据恢复，步骤如下：

1. 修改 values.yaml：

    - 将 clusterName 设置为目标 TiDB 集群名；
    - 将 mode 设置为 restore；
    - 将 name 设置为用于恢复的备份名字（你可以参考查看备份来寻找可用的备份数据）。假如备份数据存储在 Google Cloud Storage，Ceph Object Storage 或 Amazon S3 中，你必须保证这些存储的相关配置与执行全量备份时一致。

2. 创建一个包含数据库用户名和密码的 Kubernetes Secret，该用户必须拥有数据备份所需的数据库相关权限，同时，将 values.yaml 中的 secretName 设置为该 Secret 的名字（默认为 backup-secret，假如你在全量备份时已经创建了该 Secret，则可以跳过这步）：

    ```
    kubectl create secret generic backup-secret -n <namespace> --from-
        ↪ literal=user=<user> --from-literal=password=<password>
    ```

3. 恢复数据：

    ```
    helm install pingcap/tidb-backup --namespace=<namespace> --name=<
        ↪ restore-name> -f values.yaml --version=<tidb-operator-version>
    ```

8.5.1.3 增量备份

增量备份使用 TiDB Binlog 工具从 TiDB 集群收集 Binlog，并提供实时备份和向其它数据库的实时同步能力。

有关 Kubernetes 上运维 TiDB Binlog 的详细指南，可参阅TiDB Binlog。

8.5.1.3.1 Pump 缩容

缩容 Pump 需要先将单个 Pump 节点从集群中下线，然后运行 helm upgrade 命令将对应的 Pump Pod 删除，并对每个节点重复上述步骤。

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（<version> 为当前 TiDB 的版本）。

```
kubectl run offline-pump-<ordinal-id> --image=pingcap/tidb-binlog:<
    ↪ version> --namespace=<namespace> --restart=OnFailure -- /
    ↪ binlogctl -pd-urls=http://<release-name>-pd:2379 -cmd offline-
    ↪ pump -node-id <release-name>-pump-<ordinal-id>:8250
```

   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：

   修改 values.yaml 文件中 binlog.pump.replicas 为 2，然后执行如下命令来删除 Pump Pod：

```
helm upgrade <release-name> pingcap/tidb-cluster -f values.yaml --
    ↪ version=<chart-version>
```

8.5.2 恢复 Kubernetes 上的 TiDB 集群数据

本文介绍了如何使用 TiDB Lightning 快速恢复 Kubernetes 上的 TiDB 集群数据。

TiDB Lightning 包含两个组件：tidb-lightning 和 tikv-importer。在 Kubernetes 上，tikv-importer 位于 TiDB 集群的 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。

为了使用 TiDB Lightning 恢复数据，tikv-importer 和 tidb-lightning 都必须分别部署。

8.5.2.1 部署 tikv-importer

tikv-importer 可以在一个现有的 TiDB 集群上启用，或者在新建 TiDB 集群时启用。

- 在新建一个 TiDB 集群时启用 tikv-importer：

  1. 在 tidb-cluster 的 values.yaml 文件中将 importer.create 设置为 true。
  2. 部署该集群。

```
helm install pingcap/tidb-cluster --name=<tidb-cluster-release-name
    ↪ > --namespace=<namespace> -f values.yaml --version=<chart-
    ↪ version>
```

- 配置一个现有的 TiDB 集群以启用 tikv-importer：

1. 在该 TiDB 集群的 `values.yaml` 文件中将 `importer.create` 设置为 `true`。
2. 升级该 TiDB 集群。

```
helm upgrade <tidb-cluster-release-name> pingcap/tidb-cluster -f
    ↪ values.yaml --version=<chart-version>
```

8.5.2.2 部署 tidb-lightning

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
```

   tidb-lightning Helm chart 支持恢复本地或远程的备份数据。

   - 本地模式

     本地模式要求 Mydumper 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 Mydumper 备份数据目录路径，该路径中需要包含名为 `metadata` 的文件。

   - 远程模式

     与本地模式不同，远程模式需要使用 rclone 将 Mydumper 备份 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)、AWS S3 和 Ceph Object Storage。

     1. 确保 `values.yaml` 中的 `dataSource.local.nodeName` 和 `dataSource.local.hostPath` 被注释掉。
     2. 新建一个包含 rclone 配置的 Secret。rclone 配置示例如下。一般只需要配置一种云存储。有关其他的云存储，请参考 rclone 官方文档。

```
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 = <my-access-key>
```

```
secret_access_key = <my-secret-key>
region = us-east-1
[ceph]
type = s3
provider = Ceph
env_auth = false
access_key_id = <my-access-key>
secret_access_key = <my-secret-key>
endpoint = <ceph-object-store-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.json> | jq -c .`
    ↪ 命令获取。
service_account_credentials = <service-account-json-file-
    ↪ content>
```

使用你的实际配置替换上述配置中的占位符，并将该文件存储为 `secret.yaml`。然后通过 `kubectl apply -f secret.yaml -n <namespace>` 命令创建该 Secret。

3. 将 `dataSource.remote.storageClassName` 设置为 Kubernetes 集群中现有的一个存储类型。

2. 部署 TiDB Lightning

```
helm install pingcap/tidb-lightning --name=<tidb-lightning-release-name
    ↪ > --namespace=<namespace> --set failFast=true -f tidb-lightning-
    ↪ values.yaml --version=<chart-version>
```

当 TiDB Lightning 未能成功恢复数据时，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：**
>
> 目前，即使数据被成功恢复，TiDB Lightning 也会报出非零错误码并退出，这会导致 Job 失败。因此，数据恢复成功与否需要通过查看 tidb-lightning pod 的日志进行确认。

如果 TiDB Lightning 未能成功恢复数据，需要采用以下步骤进行手动干预：

1. 运行 `kubectl delete job -n <namespace> <tidb-lightning-release-name>-tidb-lightning`，删除 lightning Job。

2. 运 行 `helm template pingcap/tidb-lightning --name <tidb-lightning-release-name> --set failFast=false -f tidb-lightning-values.yaml | kubectl apply -n <namespace> -f -`，重新创建禁用 `failFast` 的 lightning Job。
3. 当 lightning pod 重新运行时，在 lightning 容器中执行 `kubectl exec -it -n <namesapce> <tidb-lightning-pod-name> sh` 命令。
4. 运行 `cat /proc/1/cmdline`，获得启动脚本。
5. 参考故障排除指南，对 lightning 进行诊断。

8.5.2.3 销毁 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：

1. 在 TiDB 集群 chart 的 `values.yaml` 文件中将 `importer.create` 设置为 `false`。
2. 然后运行 `helm upgrade <tidb-cluster-release-name> pingcap/tidb-cluster -f values.yaml`。

删除 tidb-lightning 的方法：

- 运行 `helm delete <tidb-lightning-release-name> --purge`。

8.6 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。

8.6.1 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，对容器中应用的日志收集可以直接在宿主机上完成。

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。

Kubernetes 官方文档中提供了 Stackdriver 的日志收集方案可供参考。

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。

如果不通过单独的日志收集工具汇总日志，你也可以直接使用 kubectl 工具查看某个容器的运行日志，但这一方法无法查看已销毁容器的日志：

```
kubectl logs -n <namespace> <tidbPodName>
```

若需查看容器重启之前的日志，可以在执行上述命令时添加 -p 参数。

如果需要从多个 Pod 获取日志，可以使用 stern：

```
stern -n <namespace> tidb -c slowlog
```

8.6.2 TiDB 慢查询日志

对于 3.0 之前的版本，在默认情况下，TiDB 会打印慢查询日志到标准输出，和应用日志混在一起。

- 如果 TiDB 版本 <= v2.1.7，你可以通过关键字 SLOW_QUERY 来筛选慢查询日志，例如：

  ```
  kubectl logs -n <namespace> <tidbPodName> | grep SLOW_QUERY
  ```

- 如果 TiDB 版本 >= v2.1.8，由于慢查询日志格式发生变化，不太方便分离慢查询日志，建议参考下面内容配置 separateSlowLog: true 单独查看慢查询日志。

在一些情况下，你可能希望使用一些工具或自动化系统对日志内容进行分析、处理。TiDB 各组件的应用日志使用了统一的日志格式以便于程序解析，但由于慢查询日志使用的是与 MySQL 兼容的多行格式，与应用日志混在一起时可能会对解析造成困难。

如果希望将慢查询日志和应用日志区分开，可以在 values.yaml 文件中配置 separateSlowLog 参数，将慢查询日志输出到一个专用的旁路容器中，这样慢查询日志在宿主机上会被输出到一个单独的文件，和应用日志分开。

修改方法为编辑 values.yaml 文件，将 separateSlowLog 参数设置为 true：

```
## Uncomment the following line to enable separate output of the slow query
    ↪ log
    separateSlowLog: true
```

之后再运行 helm upgrade 使配置生效，然后可以通过名为 slowlog 的 sidecar 容器查看慢查询日志：

```
kubectl logs -n <namespace> <tidbPodName> -c slowlog
```

对于 3.0 及更新的版本，TiDB 将慢查询日志输出到独立的 slowlog.log 文件中，并且 separateSlowLog 是默认开启的，所以可以直接通过 sidecar 容器查看慢查询日志，无需额外设置。

> 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。

8.6.3 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。

8.7 Kubernetes 上的 TiDB 集群故障自动转移

故障自动转移是指在 TiDB 集群的某些节点出现故障时，TiDB Operator 会自动添加一个节点，保证 TiDB 集群的高可用，类似于 K8s 的 Deployment 行为。

由于 TiDB Operator 基于 StatefulSet 来管理 Pod，但 StatefulSet 在某些 Pod 发生故障时不会自动创建新节点来替换旧节点，所以，TiDB Operator 扩展了 StatefulSet 的这种行为，添加了 Auto Failover 功能。

Auto Failover 功能在 TiDB Operator 中默认开启。部署 TiDB Operator 时，可通过设置 charts/tidb-operator/values.yaml 文件的 controllerManager.autoFailover 为 false 关闭该功能：

```
controllerManager:
 serviceAccount: tidb-controller-manager
 logLevel: 2
 replicas: 1
 resources:
   limits:
```

```
    cpu: 250m
    memory: 150Mi
  requests:
    cpu: 80m
    memory: 50Mi
# 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
```

pdFailoverPeriod、tikvFailoverPeriod 和 tidbFailoverPeriod 默认均为 5 分钟，它们的含义是在确认实例故障后的等待超时时间，超过这个时间后，TiDB Operator 就开始做自动的故障转移。

8.7.1 实现原理

TiDB 集群有 PD、TiKV 和 TiDB 三个组件，它们的故障转移策略有所不同，本节将详细介绍这三种策略。

8.7.1.1 PD 故障转移策略

假设 PD 集群有 3 个节点，如果一个 PD 节点挂掉超过 5 分钟（pdFailoverPeriod 可配置），TiDB Operator 首先会将这个 PD 节点下线，然后再添加一个新的 PD 节点。此时会有 4 个 Pod 同时存在，待挂掉的 PD 节点恢复后，TiDB Operator 会将新启动的节点删除掉，恢复成原来的 3 个节点。

8.7.1.2 TiKV 故障转移策略

当一个 TiKV 节点无法正常工作后，该节点的状态会变为 Disconnected，30 分钟（通过 pd.config 文件中 [schedule] 部分的 max-store-down-time = "30m" 来配置）后会变成 Down 状态，TiDB Operator 会在此基础上再等待 5 分钟（tikvFailoverPeriod 可配置），如果该 TiKV 节点仍不能恢复，就会新起一个 TiKV 节点。待挂掉的 TiKV 节点恢复后，TiDB Operator 不会自动删除新起的节点，用户需要手动减少 TiKV 节点，恢复成原来的节点数。操作方法是将该 TiKV 节点从 TidbCluster 对象的 status.tikv.failureStores 字段中删除：

```
kubectl edit tc -n <namespace> <clusterName>
```

```
...
status
```

```
  tikv:
    failureStores:
      "1":
        podName: cluster1-tikv-0
        storeID: "1"
      "2":
        podName: cluster1-tikv-1
        storeID: "2"
...
```

cluster1-tikv-0 节点恢复后，将其删除后变为：

```
...
status
  tikv:
    failureStores:
      "2":
        podName: cluster1-tikv-1
        storeID: "2"
...
```

8.7.1.3 TiDB 故障转移策略

假设 TiDB 集群有 3 个节点，TiDB 的故障转移策略跟 Kubernetes 中的 Deployment 的是一致的。如果一个 TiDB 节点挂掉超过 5 分钟（tidbFailoverPeriod 可配置），TiDB Operator 会添加一个新的 TiDB 节点。此时会有 4 个 Pod 同时存在，待挂掉的 TiDB 节点恢复后，TiDB Operator 会将新启动的节点删除掉，恢复成原来的 3 个节点。

9 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。

9.1 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。

9.1.1 水平扩缩容操作

1. 修改集群的 value.yaml 文件中的 pd.replicas、tidb.replicas、tikv.replicas 至期望值。

2. 执行 `helm upgrade` 命令进行扩缩容：

```
helm upgrade <release-name> pingcap/tidb-cluster -f values.yaml --
    ↪ version=<chart-version>
```

3. 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 组件在滚动升级的过程中不会触发扩缩容操作。
- TiKV 组件在缩容过程中会调用 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 tidbcluster -n <namespace> <release-name> -o json | jq '.status.tikv.stores'` 查看 TiKV 是否处于下线 `Offline` 状态。
- PD、TiKV 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。
- 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。

9.2 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。

9.2.1 垂直扩缩容操作

1. 修改 `values.yaml` 文件中的 `tidb.resources`、`tikv.resources`、`pd.resources` 至期望值。

2. 执行 `helm upgrade` 命令进行升级：

```
helm upgrade <release-name> pingcap/tidb-cluster -f values.yaml --
    ↪ version=<chart-version>
```

3. 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，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，由于 PD、TiKV 使用了 Local PV，升级后还需要调度回原节点，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。

10 升级

10.1 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。

10.1.1 升级 TiDB 版本

1. 修改集群的 values.yaml 文件中的 tidb.image、tikv.image、pd.image 的值为新版本镜像；

2. 执行 `helm upgrade` 命令进行升级：

```
helm upgrade <release-name> pingcap/tidb-cluster -f values.yaml --
    ↪ version=<chart-version>
```

3. 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。

10.1.2 更新 TiDB 集群配置

默认条件下，修改配置文件不会自动应用到 TiDB 集群中，只有在实例重启时，才会重新加载新的配置文件。

您可以开启配置文件自动更新特性，在每次配置文件更新时，自动执行滚动更新，将修改后的配置应用到集群中。操作步骤如下：

1. 修改集群的 `values.yaml` 文件，将 `enableConfigMapRollout` 的值设为 `true`；
2. 根据需求修改 `values.yaml` 中需要调整的集群配置项；
3. 执行 `helm upgrade` 命令进行升级：

```
helm upgrade <release-name> pingcap/tidb-cluster -f values.yaml --
    ↪ version=<chart-version>
```

4. 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。

注意：

- 将 `enableConfigMapRollout` 特性从关闭状态打开时，即使没有配置变更，也会触发一次 PD、TiKV、TiDB 的滚动更新。

10.1.3 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`（TiDB Operator 版本 > v1.0.0-beta.3 ）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 <release-name> -n <namespace> tidb.pingcap.
    ↪ com/force-upgrade=true
```

然后执行对应操作中的 `helm upgrade` 命令：

```
helm upgrade <release-name> pingcap/tidb-cluster -f values.yaml --version=<
    ↪ chart-version>
```

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 <release-name> -n <namespace> tidb.pingcap.
    ↪ com/force-upgrade-
```

10.2 升级 TiDB Operator

本文介绍如何升级 TiDB Operator。

10.2.1 升级步骤

1. 更新 CRD (Custom Resource Definition)：

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/<version>/manifests/crd.yaml && \
kubectl get crd tidbclusters.pingcap.com
```

注意：

`<version>` 在后续文档中代表 TiDB Operator 版本，例如 `v1.1.0`，可以通过 `helm search -l tidb-operator` 查看当前支持的版本。

2. 获取你要安装的 tidb-operator chart 中的 values.yaml 文件：

```
mkdir -p /home/tidb/tidb-operator/<version> && \
helm inspect values pingcap/tidb-operator --version=<version> > /home/
    ↪ tidb/tidb-operator/<version>/values-tidb-operator.yaml
```

3. 修改 /home/tidb/tidb-operator/<version>/values-tidb-operator.yaml 中 operatorImage 镜像版本，并将旧版本 values.yaml 中的自定义配置合并到 /home/tidb/tidb-operator/<version>/values-tidb-operator.yaml，然后执行 helm upgrade：

```
helm upgrade tidb-operator pingcap/tidb-operator --version=<version> -f
    ↪  /home/tidb/tidb-operator/<version>/values-tidb-operator.yaml
```

> 注意：
> TiDB Operator 升级之后，所有 TiDB 集群中的 discovery deployment 都会自动升级到指定的 TiDB Operator 版本。

10.2.2 升级 Kubernetes

当你的 Kubernetes 集群有版本升级，请确保 kubeSchedulerImageTag 与之匹配。默认情况下，这个值是由 Helm 在安装或者升级过程中生成的，要修改它你需要执行 helm ↪ upgrade。

11 工具

11.1 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。

11.1.1 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。

11.1.1.1 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。

11.1.1.2 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
```

11.1.2 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
```

11.1.3 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。

11.1.4 所有命令

11.1.4.1 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
```

11.1.4.2 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
```

11.1.4.3 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
```

11.1.4.4 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
```

11.1.4.5 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
```

11.1.4.6 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`
–container	-c	选择需要诊断的容器，默认为 Pod 定义中的第一个容器
–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。

11.1.4.6.1 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
```

11.1.4.6.2 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。

11.1.4.7 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
```

11.1.4.8 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
```

11.1.4.9 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

11.2 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。

11.2.1 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
```

11.2.2 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> <tikv-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，再使用 tkctl debug 命令在目标 TiKV Pod 中启动一个包含 tikv-ctl 可执行文件的新容器来执行操作，步骤如下：

  1. 进入诊断模式：

     ```
     kubectl annotate pod <tikv-pod-name> -n <namespace> runmode=debug
     ```

  2. 关闭 TiKV 进程：

     ```
     kubectl exec <tikv-pod-name> -n <namespace> -c tikv -- kill -s TERM
         ↪ 1
     ```

  3. 启动 debug 容器：

     ```
     tkctl debug <tikv-pod-name> -c tikv
     ```

  4. 开始使用 tikv-ctl 的本地模式，需要注意的是 tikv 容器的根文件系统在 /proc/1/root 下，因此执行命令时也需要调整数据目录的路径：

     ```
     tikv-ctl --db /path/to/tikv/db size -r 2
     ```

     Kubernetes 上 TiKV 实例在 debug 容器中的的默认 db 路径是 /proc/1/root/
     ↪ var/lib/tikv/db size -r 2

11.2.3 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> <tidb-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
```

11.2.4 使用 Helm

Helm 是一个 Kubernetes 的包管理工具，确保安装的 Helm 版本为 2.11.0 ≤ Helm < 2.16.4。安装步骤如下：

1. 参考官方文档安装 Helm 客户端
2. 安装 Helm 服务端

   在集群中应用 Helm 服务端组件 `tiller` 所需的 `RBAC` 规则，并安装 `tiller`：

   ```
   kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
       ↪ operator/master/manifests/tiller-rbac.yaml && \
   helm init --service-account=tiller --upgrade
   ```

   如果无法访问 gcr.io，你可以尝试 mirror 仓库：

   ```
   helm init --service-account=tiller --upgrade --tiller-image registry.cn
       ↪ -hangzhou.aliyuncs.com/google_containers/tiller:$(helm version --
       ↪ client --short | grep -Eo 'v[0-9]\.[0-9]+\.[0-9]+')
   ```

   通过下面命令确认 `tiller` Pod 进入 running 状态：

   ```
   kubectl get po -n kube-system -l name=tiller
   ```

   如果 Kubernetes 集群没有启用 `RBAC`，那么可以直接使用下列命令安装 `tiller`：

   ```
   helm init --upgrade
   ```

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 pingcap -l
```

```
NAME                  CHART VERSION APP VERSION DESCRIPTION
pingcap/tidb-backup   v1.0.0                   A Helm chart for TiDB Backup
    ↪ or Restore
pingcap/tidb-cluster v1.0.0                   A Helm chart for TiDB Cluster
pingcap/tidb-operator v1.0.0                  tidb-operator Helm chart for
    ↪ Kubernetes
```

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
```

Helm 的常用操作有部署（helm install）、升级（helm upgrade）、销毁（helm del）、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，我们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 <chart-name> --name=<release-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 del --purge <release-name>
```

更多 helm 的相关文档，请参考 Helm 官方文档。

11.2.5 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。

12 组件

12.1 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。

12.1.1 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，TiDB Scheduler 实现了如下几种自定义的调度规则。

12.1.1.1 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

PD 集群规模（Replicas）	每个节点最多可调度的 PD 实例数量
4	1
5	2
…	

12.1.1.2 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
…		

12.1.1.3 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。

12.1.2 工作原理

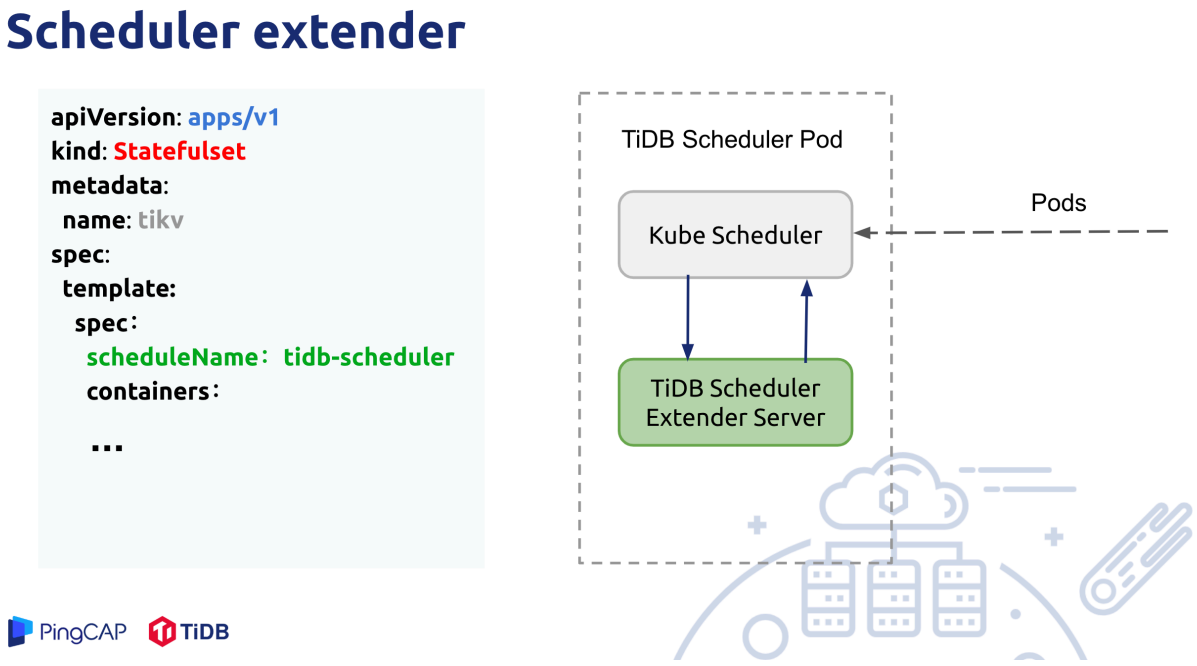


Figure 13: TiDB Scheduler 工作原理

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。

13 Kubernetes 上的 TiDB 集群故障诊断

本文介绍了 Kubernetes 上 TiDB 集群的一些常见故障以及诊断解决方案。

13.1 诊断模式

当 Pod 处于 `CrashLoopBackoff` 状态时，Pod 内会容器不断退出，导致无法正常使用 `kubectl exec` 或 `tkctl debug`，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` 或 `tkctl debug` 连接 Pod 内的容器进行诊断。

操作方式：

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。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
```

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。

13.2 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 delete` 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 pingcap/tidb-cluster -n <release-name> --namespace=<namespace>
    ↪ --version=<chart-version> -f values.yaml
```

13.3 Pod 未正常创建

通过 `helm install` 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 get statefulsets -n <namespace>
kubectl describe statefulsets -n <namespace> <release-name>-pd
```

13.4 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 <release-name>-pd
   kubectl -n <namespace> get endpoints <release-name>-tidb
   kubectl -n <namespace> get endpoints <release-name>-pd-peer
   kubectl -n <namespace> get endpoints <release-name>-tikv-peer
   kubectl -n <namespace> get endpoints <release-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 ClusterIP）：

   ```
   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
```

13.5 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、Monitor 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>
```

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。

如果是 PVC 的 StorageClass 找不到，需要在 values.yaml 里面将 storageClassName 修改为集群中可用的 StorageClass 名字，执行 helm upgrade，然后将 Statefulset 删除，并且将对应的 PVC 也都删除，可以通过以下命令获取集群中可用的 StorageClass：

```
kubectl get storageclass
```

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。

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。

13.6 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）。可能导致 CrashLoopBackOff 的原因有很多，此时，最有效的定位办法是查看 Pod 内容器的日志：

```
kubectl -n <namespace> logs -f <pod-name>
```

假如本次日志没有能够帮助诊断的有效信息，可以添加 -p 参数输出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。

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。

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
```

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。

13.7 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>
```

检查 TiDB 组件的日志，看日志是否有报错。

```
kubectl logs -f <tidb-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 service domain 或 clusterIP 访问 TiDB 服务，假如 serviceName 或 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 service domain 或 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 service domain 或 clusterIP 到 PodIP 之间的连接上，排查项如下：

    - 检查 DNS 服务是否正常：

    ```
    kubectl get po -n kube-system -l k8s-app=kube-dns
    dig <tidb-service-domain>
    ```

    - 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

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

13.8 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 <release-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=<
        ↪ release-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。

13.9 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。

14 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。

14.1 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：

- 第一次部署集群

  在 TiDB 集群的 `values.yaml` 文件中，修改 `timezone` 配置，例如：`timezone: Asia/Shanghai`，然后部署 TiDB 集群。

- 集群已经在运行

  如果 TiDB 集群已经在运行，需要做如下修改：

  - 在 TiDB 集群的 `values.yaml` 文件中，修改 `timezone` 配置，例如：`timezone: Asia/Shanghai`，然后升级 TiDB 集群。
  - 参考时区支持，修改 TiDB 服务时区配置。

14.2 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。

14.3 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。

14.4 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。

14.5 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。

14.6 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 <pd-pod-name> -n <namespace> -- cat /etc/pd/pd.toml
```

- 查看 TiKV 配置文件

```
kubectl exec -it <tikv-pod-name> -n <namespace> -- cat /etc/tikv/tikv.
    ↪ toml
```

- 查看 TiDB 配置文件

```
kubectl exec -it <tidb-pod-name> -c tidb -n <namespace> -- cat /etc/
    ↪ tidb/tidb.toml
```

14.7 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
```

15 版本发布历史

15.1 v1.0

15.1.1 TiDB Operator 1.0.7 Release Notes

Release date: June 16, 2020

TiDB Operator version: 1.0.7

15.1.1.1 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)

15.1.2 TiDB Operator 1.0.6 Release Notes

Release date: December 27, 2019

TiDB Operator version: 1.0.6

15.1.2.1 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
...
```

15.1.2.1.1 Monitor

- Enable alert rule persistence (#898)
- Add node & pod info in TiDB Grafana (#885)

15.1.2.1.2 TiDB Scheduler

- Refine scheduler error messages (#1373)

15.1.2.1.3 Compatibility

- Fix the compatibility issue in Kubernetes v1.17 (#1241)
- Bind the `system:kube-scheduler` ClusterRole to the `tidb-scheduler` service account (#1355)

15.1.2.1.4 TiKV Importer

- Fix the default `tikv-importer` configuration (#1415)

15.1.2.1.5 E2E

- Ensure pods unaffected when upgrading (#955)

15.1.2.1.6 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)

15.1.3 TiDB Operator 1.0.5 Release Notes

Release date: December 11, 2019

TiDB Operator version: 1.0.5

15.1.3.1 v1.0.5 What's New

There is no action required if you are upgrading from v1.0.4.

15.1.3.1.1 Scheduled Backup

- Fix the issue that backup failed when `clusterName` is too long (#1229)

15.1.3.1.2 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)

15.1.3.1.3 Compatibility

- Fix `tidb-scheduler` RBAC permission in Kubernetes v1.16 (#1282)
- Do not set `DNSPolicy` if `hostNetwork` is disabled to keep backward compatibility (#1287)

15.1.3.1.4 E2E

- Fix e2e nil point dereference (#1221)

15.1.4 TiDB Operator 1.0.4 Release Notes

Release date: November 23, 2019

TiDB Operator version: 1.0.4

15.1.4.1 v1.0.4 What's New

15.1.4.1.1 Action Required

There is no action required if you are upgrading from v1.0.3.

15.1.4.1.2 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.

15.1.4.1.3 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)

15.1.4.2 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)

15.1.4.3 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)

15.1.5 TiDB Operator 1.0.3 Release Notes

Release date: November 13, 2019

TiDB Operator version: 1.0.3

15.1.5.1 v1.0.3 What's New

15.1.5.1.1 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`.

15.1.5.1.2 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

15.1.5.2 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)

15.1.6 TiDB Operator 1.0.2 Release Notes

Release date: November 1, 2019

TiDB Operator version: 1.0.2

15.1.6.1 v1.0.2 What's New

15.1.6.1.1 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.

15.1.6.1.2 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

15.1.6.1.3 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

15.1.6.2 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)

15.1.7 TiDB Operator 1.0.1 Release Notes

Release date: September 17, 2019

TiDB Operator version: 1.0.1

15.1.7.1 v1.0.1 What's New

15.1.7.1.1 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.

15.1.7.1.2 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`

15.1.7.1.3 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

15.1.7.2 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)

15.1.8 TiDB Operator 1.0 GA Release Notes

Release date: July 30, 2019

TiDB Operator version: 1.0.0

15.1.8.1 v1.0.0 What's New

15.1.8.1.1 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.

15.1.8.1.2 Stability Test Cases Added

- Stop all etcds and kubelets

15.1.8.1.3 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

15.1.8.1.4 Bug Fixes

- Fix sysbench installation on bastion machine of AWS deployment
- Fix TiKV metrics monitoring in default setup

15.1.8.2 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)

15.1.9 TiDB Operator 1.0 RC.1 Release Notes

Release date: July 12, 2019

TiDB Operator version: 1.0.0-rc.1

15.1.9.1 v1.0.0-rc.1 What's New

15.1.9.1.1 Stability test cases added

- Stop kube-proxy
- Upgrade tidb-operator

15.1.9.1.2 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.

15.1.9.1.3 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

15.1.9.2 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)

15.1.10 TiDB Operator 1.0 Beta.3 Release Notes

Release date: June 6, 2019

TiDB Operator version: 1.0.0-beta.3

15.1.10.1 v1.0.0-beta.3 What's New

15.1.10.1.1 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.

15.1.10.1.2 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

15.1.10.1.3 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

15.1.10.1.4 Documentation Improvement

- GCP one-command deployment
- Refine user guides
- Improve GKE, AWS, Aliyun guide

15.1.10.1.5 Pass User Acceptance Tests

15.1.10.1.6 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

15.1.10.2 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)

15.1.11 TiDB Operator 1.0 Beta.2 Release Notes

Release date: May 10, 2019

TiDB Operator version: 1.0.0-beta.2

15.1.11.1 v1.0.0-beta.2 What's New

15.1.11.1.1 Stability has been greatly enhanced

- Refactored e2e test
- Added stability test, 7x24 running

15.1.11.1.2 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

15.1.11.1.3 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

15.1.11.1.4 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

15.1.11.2 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)

15.1.12 TiDB Operator 1.0 Beta.1 P2 Release Notes

Release date: February 21, 2019

TiDB Operator version: 1.0.0-beta.1-p2

15.1.12.1 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)

15.1.13 TiDB Operator 1.0 Beta.1 P1 Release Notes

Release date: January 7, 2019

TiDB Operator version: 1.0.0-beta.1-p1

15.1.13.1 Bug Fixes

- Fix scheduler policy issue, works on kubernetes v1.10, v1.11 and v1.12 now (#256)

15.1.13.2 Docs

- Proposal: add multiple statefulsets support to TiDB Operator (#240)
- Update roadmap (#258)

15.1.14 TiDB Operator 1.0 Beta.1 Release Notes

Release date: December 27, 2018

TiDB Operator version: 1.0.0-beta.1

15.1.14.1 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)

15.1.14.2 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)

15.1.15 TiDB Operator 1.0 Beta.0 Release Notes

Release date: November 26, 2018

TiDB Operator version: 1.0.0-beta.0

15.1.15.1 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)

15.2 v0

15.2.1 TiDB Operator 0.4 Release Notes

Release date: November 9, 2018

TiDB Operator version: 0.4.0

15.2.1.1 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)

15.2.1.2 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)

15.2.2 TiDB Operator 0.3.1 Release Notes

Release date: October 31, 2018

TiDB Operator version: 0.3.1

15.2.2.1 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)

15.2.2.2 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)

15.2.3 TiDB Operator 0.3.0 Release Notes

Release date: October 12, 2018

TiDB Operator version: 0.3.0

15.2.3.1 Notable Changes

- Add full backup support
- Add TiDB Binlog support
- Add graceful upgrade feature
- Allow monitor data to be persistent

15.2.4 TiDB Operator 0.2.1 Release Notes

Release date: September 20, 2018

TiDB Operator version: 0.2.1

15.2.4.1 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)

15.2.5 TiDB Operator 0.2.0 Release Notes

Release date: September 11, 2018

TiDB Operator version: 0.2.0

15.2.5.1 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

15.2.6 TiDB Operator 0.1.0 Release Notes

Release date: August 22, 2018

TiDB Operator version: 0.1.0

15.2.6.1 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
