TiDB in Kubernetes Documentation

PingCAP Inc.

20230320

Table of Contents

1 TiDB in Kubernetes Docs **9**

2 TiDB Operator Overview **9**

2.1 TiDB Operator architecture · · · 10

2.2 Manage TiDB clusters using TiDB Operator · · · 11

3 Benchmark **13**

3.1 TiDB in Kubernetes Sysbench Performance Test · · · 13

3.1.1 Test purpose · · · 13

3.1.2 Test environment · · · 13

3.1.3 Test report · · · 17

3.1.4 Conclusion · · · 34

4 Get Started **35**

4.1 Deploy TiDB on Google Cloud · · · 35

4.1.1 Select a project · · · 35

4.1.2 Enable API access · · · 36

4.1.3 Configure gcloud defaults · · · 36

4.1.4 Launch a 3-node Kubernetes cluster · · · 36

4.1.5 Install Helm · · · 36

4.1.6 Add Helm repo · · · 37

4.1.7 Deploy TiDB Operator · · · 37

4.1.8 Deploy your first TiDB cluster 38
4.1.9 Connect to the TiDB cluster 38
4.1.10 Scale out the TiDB cluster 39
4.1.11 Accessing the Grafana dashboard 39
4.1.12 Destroy the TiDB cluster 40
4.1.13 Shut down the Kubernetes cluster 40
4.2 Deploy TiDB in the Minikube Cluster 40
4.2.1 Start a Kubernetes cluster with minikube 40
4.2.2 Install TiDB Operator and run a TiDB cluster 41
4.2.3 FAQs 45

5 Deploy 45
5.1 Prerequisites for TiDB in Kubernetes 45
5.1.1 Software version 45
5.1.2 The configuration of kernel parameters 45
5.1.3 Hardware and deployment requirements 48
5.1.4 Kubernetes requirements for system resources 48
5.1.5 TiDB cluster's requirements for resources 49
5.1.6 A case of planning TiDB clusters 49
5.2 Deploy TiDB Operator in Kubernetes 50
5.2.1 Prerequisites 51
5.2.2 Deploy Kubernetes cluster 51
5.2.3 Install Helm 52
5.2.4 Configure local persistent volume 52
5.2.5 Install TiDB Operator 52
5.2.6 Customize TiDB Operator 53
5.3 Deploy TiDB on General Kubernetes 53
5.3.1 Prerequisites 53
5.3.2 Configure TiDB cluster 54
5.3.3 Deploy TiDB Cluster 55

5.4 Deploy TiDB on AWS EKS · · · 56
5.4.1 Prerequisites · · · 56
5.4.2 Deploy · · · 57
5.4.3 Access the database · · · 58
5.4.4 Monitor · · · 59
5.4.5 Upgrade · · · 59
5.4.6 Scale · · · 60
5.4.7 Customize · · · 60
5.4.8 Manage multiple TiDB clusters · · · 62
5.4.9 Manage the infrastructure only · · · 64
5.4.10 Destroy clusters · · · 65
5.4.11 Manage multiple Kubernetes clusters · · · 65
5.5 Deploy TiDB on GCP GKE · · · 68
5.5.1 Prerequisites · · · 69
5.5.2 Configure · · · 69
5.5.3 Deploy a TiDB cluster · · · 71
5.5.4 Access the TiDB database · · · 72
5.5.5 Interact with the GKE cluster · · · 73
5.5.6 Upgrade the TiDB cluster · · · 73
5.5.7 Manage multiple TiDB clusters · · · 74
5.5.8 Scale the TiDB cluster · · · 75
5.5.9 Customize · · · 76
5.5.10 Destroy a TiDB cluster · · · 80
5.5.11 Manage multiple Kubernetes clusters · · · 81
5.6 Deploy TiDB on Alibaba Cloud Kubernetes · · · 85
5.6.1 Prerequisites · · · 85
5.6.2 Overview of things to create · · · 86
5.6.3 Deploy · · · 86
5.6.4 Access the database · · · 87
5.6.5 Monitor · · · 88
5.6.6 Upgrade · · · 88
5.6.7 Scale · · · 88

5.6.8 Configure · · · 89
5.6.9 Manage multiple TiDB clusters · · · 89
5.6.10 Manage multiple Kubernetes clusters · · · 93
5.6.11 Destroy · · · 95
5.6.12 Limitation · · · 96
5.7 Access the TiDB Cluster in Kubernetes · · · 96
5.7.1 NodePort · · · 96
5.7.2 LoadBalancer · · · 97
5.8 Maintain TiDB Binlog · · · 97
5.8.1 Prerequisites · · · 98
5.8.2 Enable TiDB Binlog of a TiDB cluster · · · 98
5.8.3 Deploy multiple drainers · · · 101

6 Configure 102
6.1 Initialize a TiDB Cluster in Kubernetes · · · 102
6.1.1 Set initial account and password · · · 102
6.1.2 Initialize SQL statements in batch · · · 104
6.2 TiDB Cluster Configurations in Kubernetes · · · 104
6.2.1 Configuration parameters · · · 104
6.2.2 Resource configuration · · · 166
6.2.3 Disaster recovery configuration · · · 166
6.3 The Backup Configuration of TiDB in Kubernetetes · · · 168
6.3.1 Configuration · · · 168
6.4 Persistent Storage Class Configuration in Kubernetes · · · 171
6.4.1 Recommended storage classes for TiDB clusters · · · 172
6.4.2 Network PV configuration · · · 172
6.4.3 Local PV configuration · · · 173
6.4.4 Disk mount examples · · · 174
6.4.5 Data safety · · · 177
6.5 TiDB Binlog Drainer Configurations in Kubernetes · · · 178
6.5.1 Configuration parameters · · · 178

7 Monitor a TiDB Cluster in Kubernetes 185

7.1 Monitor the TiDB cluster · · · 186
7.1.1 View the monitoring dashboard · · · 186
7.1.2 Access the monitoring data · · · 186
7.2 Monitor the Kubernetes cluster · · · 187
7.2.1 Monitor the host · · · 187
7.2.2 Monitor Kubernetes components · · · 187
7.3 Alert configuration · · · 188
7.3.1 Alerts in the TiDB Cluster · · · 188
7.3.2 Alerts in Kubernetes · · · 188

8 Maintain **188**
8.1 Destroy TiDB Clusters in Kubernetes · · · 188
8.2 Restart a TiDB Cluster in Kubernetes · · · 189
8.2.1 Forcibly restart a Pod · · · 190
8.2.2 Forcibly restart all Pods of a component · · · 190
8.2.3 Forcibly restart all Pods of the TiDB cluster · · · 190
8.3 Maintain Kubernetes Nodes that Hold the TiDB Cluster · · · 191
8.3.1 Prerequisites · · · 191
8.3.2 Maintain nodes that hold PD and TiDB instances · · · 191
8.3.3 Maintain nodes that hold TiKV instances · · · 192
8.4 Use PD Recover to Recover the PD Cluster · · · 195
8.4.1 Download PD Recover · · · 195
8.4.2 Recover the PD cluster · · · 195
8.5 Backup and Restore · · · 198
8.5.1 Backup and Restore · · · 198
8.5.2 Restore Data into TiDB in Kubernetes · · · 201
8.6 Collect TiDB Logs in Kubernetes · · · 205
8.6.1 Collect logs of TiDB components in Kubernetes · · · 205
8.6.2 Collect TiDB slow query logs · · · 206
8.6.3 Collect system logs · · · 207
8.7 Automatic Failover · · · 207
8.7.1 Automatic failover policies · · · 208

9 Scale TiDB in Kubernetes **209**
9.1 Horizontal scaling · · · 209
9.1.1 Horizontal scaling operations · · · 210
9.2 Vertical scaling · · · 211
9.2.1 Vertical scaling operations · · · 211

10 Upgrade **211**
10.1 Perform a Rolling Update to a TiDB Cluster in Kubernetes · · · 211
10.1.1 Upgrade the version of TiDB cluster · · · 212
10.1.2 Change the configuration of TiDB cluster · · · 212
10.1.3 Force an upgrade of TiDB cluster · · · 213
10.2 Upgrade TiDB Operator and Kubernetes · · · 213
10.2.1 Upgrade TiDB Operator · · · 214
10.2.2 Upgrade Kubernetes · · · 214

11 Tools **214**
11.1 TiDB Kubernetes Control User Guide · · · 214
11.1.1 Installation · · · 214
11.1.2 Commands · · · 216
11.2 Tools in Kubernetes · · · 223
11.2.1 Use PD Control in Kubernetes · · · 223
11.2.2 Use TiKV Control in Kubernetes · · · 224
11.2.3 Use TiDB Control in Kubernetes · · · 225
11.2.4 Use Helm · · · 225
11.2.5 Use Terraform · · · 227

12 Components **228**
12.1 TiDB Scheduler · · · 228
12.1.1 TiDB cluster scheduling requirements · · · 228
12.1.2 How TiDB Scheduler works · · · 230

13 Troubleshoot TiDB in Kubernetes **231**

13.1 Use the diagnostic mode · · · 231
13.2 Recover the cluster after accidental deletion · · · 232
13.3 Pod is not created normally · · · 232
13.4 Network connection failure between Pods · · · 232
13.5 The Pod is in the Pending state · · · 233
13.6 The Pod is in the `CrashLoopBackOff` state · · · 234
13.7 Unable to access the TiDB service · · · 235
13.8 TiKV Store is in `Tombstone` status abnormally · · · 236
13.9 Long queries are abnormally interrupted in TiDB · · · 238

14 TiDB FAQs in Kubernetes **238**

14.1 How to modify time zone settings？ · · · 239
14.2 Can HPA or VPA be configured on TiDB components? · · · 239
14.3 What scenarios require manual intervention when I use TiDB Operator to orchestrate a TiDB cluster? · · · 239
14.4 What is the recommended deployment topology when I use TiDB Operator to orchestrate a TiDB cluster on a public cloud? · · · 239
14.5 Does TiDB Operator support TiSpark? · · · 240
14.6 How to check the configuration of the TiDB cluster? · · · 240
14.7 Why does TiDB Operator fail to schedule Pods when I deploy the TiDB clusters? · · · 240

15 Release Notes **241**

15.1 v1.0 · · · 241
15.1.1 TiDB Operator 1.0.7 Release Notes · · · 241
15.1.2 TiDB Operator 1.0.6 Release Notes · · · 241
15.1.3 TiDB Operator 1.0.5 Release Notes · · · 243
15.1.4 TiDB Operator 1.0.4 Release Notes · · · 244
15.1.5 TiDB Operator 1.0.3 Release Notes · · · 245
15.1.6 TiDB Operator 1.0.2 Release Notes · · · 246
15.1.7 TiDB Operator 1.0.1 Release Notes · · · 248
15.1.8 TiDB Operator 1.0 GA Release Notes · · · 250
15.1.9 TiDB Operator 1.0 RC.1 Release Notes · · · 254
15.1.10 TiDB Operator 1.0 Beta.3 Release Notes · · · 255

15.1.11 TiDB Operator 1.0 Beta.2 Release Notes · · · 258
15.1.12 TiDB Operator 1.0 Beta.1 P2 Release Notes · · · 261
15.1.13 TiDB Operator 1.0 Beta.1 P1 Release Notes · · · 262
15.1.14 TiDB Operator 1.0 Beta.1 Release Notes · · · 262
15.1.15 TiDB Operator 1.0 Beta.0 Release Notes · · · 263
15.2 v0 · · · 263
15.2.1 TiDB Operator 0.4 Release Notes · · · 263
15.2.2 TiDB Operator 0.3.1 Release Notes · · · 264
15.2.3 TiDB Operator 0.3.0 Release Notes · · · 265
15.2.4 TiDB Operator 0.2.1 Release Notes · · · 265
15.2.5 TiDB Operator 0.2.0 Release Notes · · · 265
15.2.6 TiDB Operator 0.1.0 Release Notes · · · 266

1 TiDB in Kubernetes Docs

2 TiDB Operator Overview

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

> **Note:**
>
> You can only deploy one TiDB Operator in a Kubernetes cluster.

The corresponding relationship between TiDB Operator and TiDB versions is as follows:

TiDB versions	Compatible TiDB Operator versions
dev	dev
TiDB >= 5.4	1.3
5.1 <= TiDB < 5.4	1.3 (Recommended), 1.2
3.0 <= TiDB < 5.1	1.3 (Recommended), 1.2, 1.1
2.1 <= TiDB < v3.0	1.0 (End of support)

2.1 TiDB Operator architecture

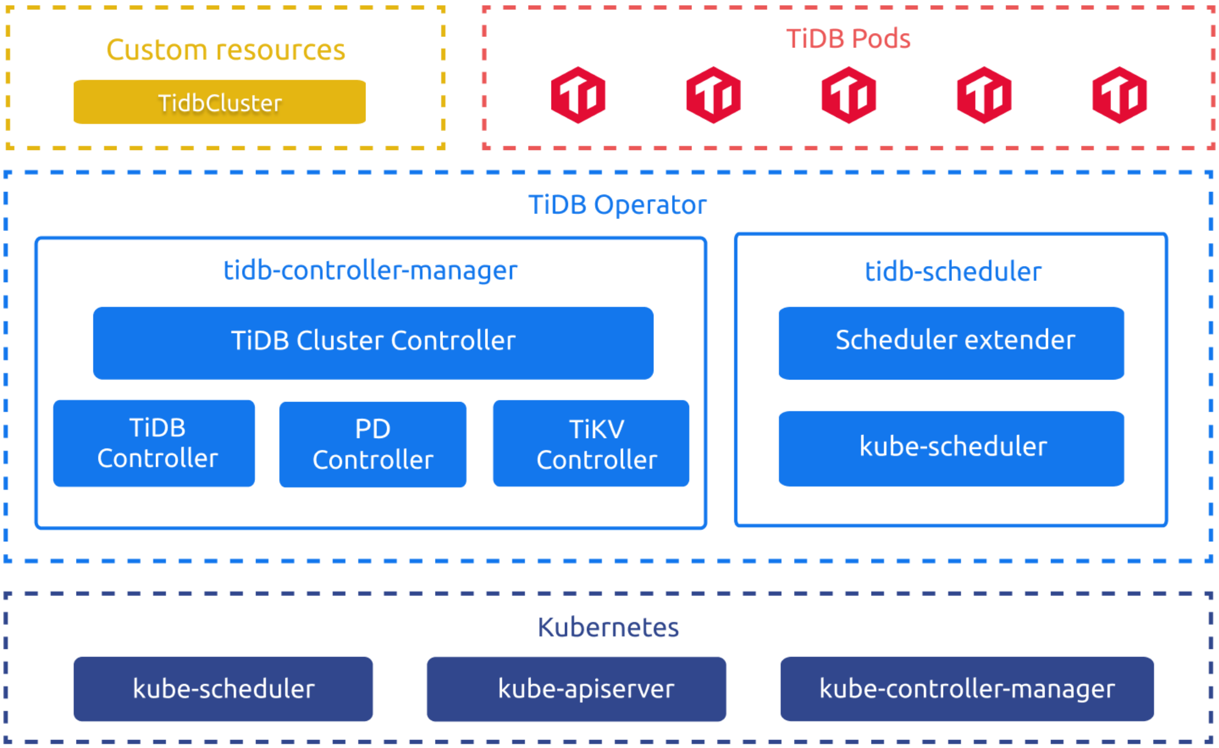


Figure 1: TiDB Operator Overview

`TidbCluster` is a custom resource defined by CRD (`CustomResourceDefinition`) and is used to describe the desired state of the TiDB cluster. The following components are responsible for the orchestration and scheduling logic in a TiDB cluster:

- `tidb-controller-manager` is a set of custom controllers in Kubernetes. These controllers constantly compare the desired state recorded in the `TidbCluster` object with the actual state of the TiDB cluster. They adjust the resources in Kubernetes to drive the TiDB cluster to meet the desired state;
- `tidb-scheduler` is a Kubernetes scheduler extension that injects the TiDB specific scheduling policies to the Kubernetes scheduler.

In addition, TiDB Operator also provides `tkctl`, the command-line interface for TiDB clusters in Kubernetes. It is used for cluster operations and troubleshooting cluster issues.

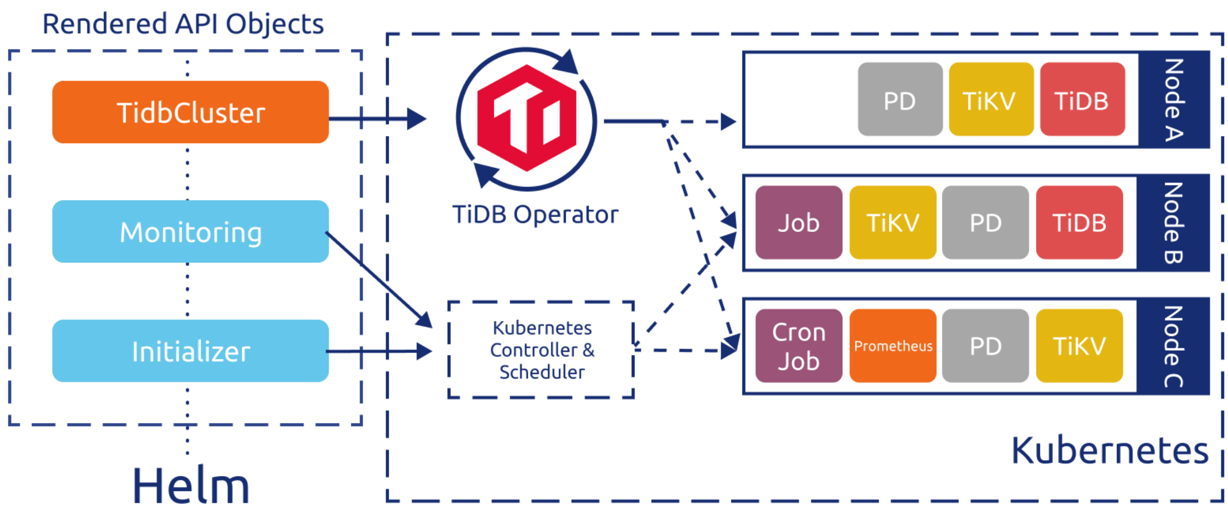


Figure 2: TiDB Operator Control Flow

The diagram above is the analysis of the control flow of TiDB Operator. Because TiDB clusters also need components such as monitoring, initialization, scheduled backup, Binlog and so on, TiDB Operator encapsulates the definition of these components in the Helm chart. The overall control process is as follows:

1. The user creates a `TidbCluster` object and a corresponding series of Kubernetes-native objects through Helm, such as a `CronJob` that performs scheduled backups;
2. TiDB Operator watches `TidbCluster` and other related objects, and constantly adjust the `StatefulSet` and `Service` objects of PD, TiKV, and TiDB based on the actual state of the cluster;
3. Kubernetes' native controller creates, updates, and deletes the corresponding `Pod` based on objects such as `StatefulSet`, `Deployment`, and `CronJob`;
4. In the `Pod` declaration of PD, TiKV, and TiDB, the `tidb-scheduler` scheduler is specified. `tidb-scheduler` applies the specific scheduling logic of TiDB when scheduling the corresponding `Pod`.

Based on the above declarative control flow, TiDB Operator automatically performs health check and fault recovery for the cluster nodes. You can easily modify the `TidbCluster` object declaration to perform operations such as deployment, upgrade and scaling.

2.2 Manage TiDB clusters using TiDB Operator

TiDB Operator provides several ways to deploy TiDB clusters in Kubernetes:

- For test environment:

- Minikube: Deploy TiDB clusters in a local Minikube environment using TiDB Operator
- GKE: Deploy TiDB clusters on GKE using TiDB Operator

- For production environment:

  - On public cloud:

    * Deploy TiDB on AWS EKS
    * Deploy TiDB on GCP GKE (beta)
    * Deploy TiDB on Alibaba Cloud ACK

  - In an existing Kubernetes cluster:

    First install TiDB Operator in a Kubernetes cluster according to Deploy TiDB Operator in Kubernetes, then deploy your TiDB clusters according to Deploy TiDB in General Kubernetes.

    You also need to adjust the configuration of the Kubernetes cluster based on Prerequisites for TiDB in Kubernetes and configure the local PV for your Kubernetes cluster to achieve low latency of local storage for TiKV according to Local PV Configuration.

Before deploying TiDB on any of the above two environments, you can always refer to TiDB Cluster Configuration Document to customize TiDB configurations.

After the deployment is complete, see the following documents to use, operate, and maintain TiDB clusters in Kubernetes:

- Access the TiDB Cluster
- Scale TiDB Cluster
- Upgrade TiDB Cluster
- Change the Configuration of TiDB Cluster
- Backup and Restore using Helm Charts
- Automatic Failover
- Monitor a TiDB Cluster in Kubernetes
- Collect TiDB Logs in Kubernetes
- Maintain Kubernetes Nodes that Hold the TiDB Cluster

When a problem occurs and the cluster needs diagnosis, you can:

- See TiDB FAQs in Kubernetes for any available solution;
- See Troubleshoot TiDB in Kubernetes to shoot troubles.

TiDB in Kubernetes provides a dedicated command-line tool `tkctl` for cluster management and auxiliary diagnostics. Meanwhile, some of TiDB's tools are used differently in Kubernetes. You can:

- Use `tkctl` according to `tkctl` Guide;
- See Tools in Kubernetes to understand how TiDB tools are used in Kubernetes.

Finally, when a new version of TiDB Operator is released, you can refer to Upgrade TiDB Operator to upgrade to the latest version.

3 Benchmark

3.1 TiDB in Kubernetes Sysbench Performance Test

Since the release of TiDB Operator GA, more users begin to deploy and manage the TiDB cluster in Kubernetes using TiDB Operator. In this report, an in-depth and comprehensive test of TiDB has been conducted on GKE, which offers insight into the influencing factors that affects the performance of TiDB in Kubernetes.

3.1.1 Test purpose

- To test the performance of TiDB on a typical public cloud platform
- To test the influences that the public cloud platform, network, CPU and different Pod networks have on the performance of TiDB

3.1.2 Test environment

3.1.2.1 Version and configuration

In this test:

- TiDB 3.0.1 and TiDB Operator 1.0.0 are used.
- Three instances are deployed for PD, TiDB and TiKV respectively.
- Each component is configured as below. Unconfigured components use the default values.

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 parameter configuration

```
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 Hardware recommendations

3.1.2.3.1 Machine types

For the test in single AZ (Available Zone), the following machine types are chosen:

Component	Instance type	Count
PD	n1-standard-4	3
TiKV	c2-standard-16	3
TiDB	c2-standard-16	3
Sysbench	c2-standard-30	1

For the test (2019.08) where the result in multiple Azs is compared with that in single AZ, the c2 machine is not simultaneously available in three AZs within the same GCP region, so the following machine types are chosen:

Component	Instance type	Count
PD	n1-standard-4	3
TiKV	n1-standard-16	3
TiDB	n1-standard-16	3
Sysbench	n1-standard-16	3

Sysbench, the pressure test platform, has a high demand on CPU in the high concurrency read test. Therefore, it is recommended that you use machines with high configuration and multiple cores so that the test platform does not become the bottleneck.

Note:

The usable machine types vary among GCP regions. In the test, disk also performs differently. Therefore, only the machines in us-central1 are applied for test.

3.1.2.3.2 Disk

The NVMe disks on GKE are still in the Alpha phase, so it requires special application to use them and is not for general usage. In this test, the iSCSI interface type is used for all local SSD disks. With reference to the official recommendations, the `discard,nobarrier` option has been added to the mounting parameter. Below is a complete example:

```
sudo mount -o defaults,nodelalloc,noatime,discard,nobarrier /dev/[
    ↪ LOCAL_SSD_ID] /mnt/disks/[MNT_DIR]
```

3.1.2.3.3 Network

GKE uses a more scalable and powerful VPC-Native mode as its network mode. In the performance comparison, TiDB is tested with Kubernetes Pod and Host respectively.

3.1.2.3.4 CPU

- In the test on single AZ cluster, the c2-standard-16 machine mode is chosen for TiDB/TiKV.
- In the comparison test on single AZ cluster and on multiple AZs cluster, the c2-standard-16 machine type cannot be simultaneously adopted in three AZs within the same GCP region, so n1-standard-16 machine type is chosen.

3.1.2.4 Operation system and parameters

GKE supports two operating systems: COS (Container Optimized OS) and Ubuntu. The Point Select test is conducted on both systems and the results are compared. Other tests are only conducted on Ubuntu.

The core is configured as below:

```
sysctl net.core.somaxconn=32768
sysctl vm.swappiness=0
sysctl net.ipv4.tcp_syncookies=0
```

The maximum number of files is configured as `1000000`.

3.1.2.5 Sysbench version and operating parameters

In this test, the version of sysbench is 1.0.17.

Before the test, the `prewarm` command of `oltp_common` is used to warm up data.

3.1.2.5.1 Initialization

```
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>` is the address of TiDB database, which is specified according to actual test needs. For example, Pod IP, Service domain name, Host IP, and Load Balancer IP (the same below).

3.1.2.5.2 Warming-up

```
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 Pressure test

```
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>` is the test case of sysbench. In this test, `oltp_point_select`, `oltp_update_index` ↪ , `oltp_update_no_index`, and `oltp_read_write` are chosen as `<test>`.

3.1.3 Test report

3.1.3.1 In single AZ

3.1.3.1.1 Pod Network vs Host Network

Kubernetes allows Pods to run in Host network mode. This way of deployment is suitable when a TiDB instance occupies the whole machine without causing any Pod conflict. The Point Select test is conducted in both modes respectively.

In this test, the operating system is 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 comparison:

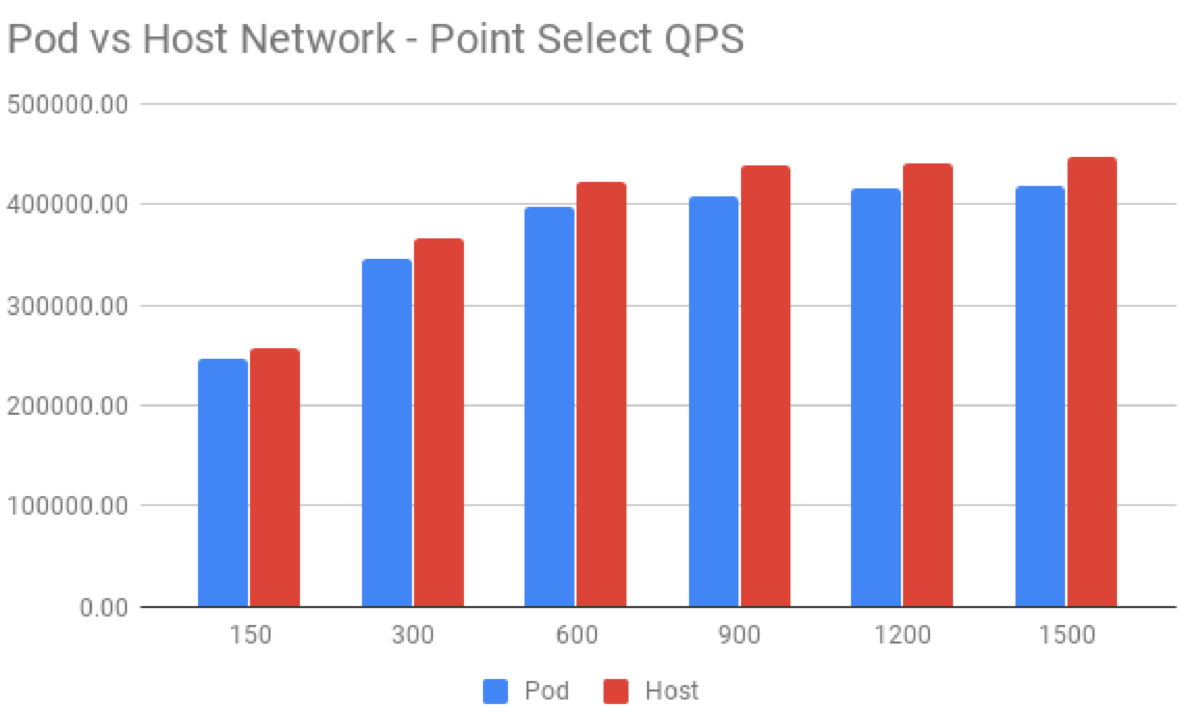


Figure 3: Pod vs Host Network

Latency comparison:

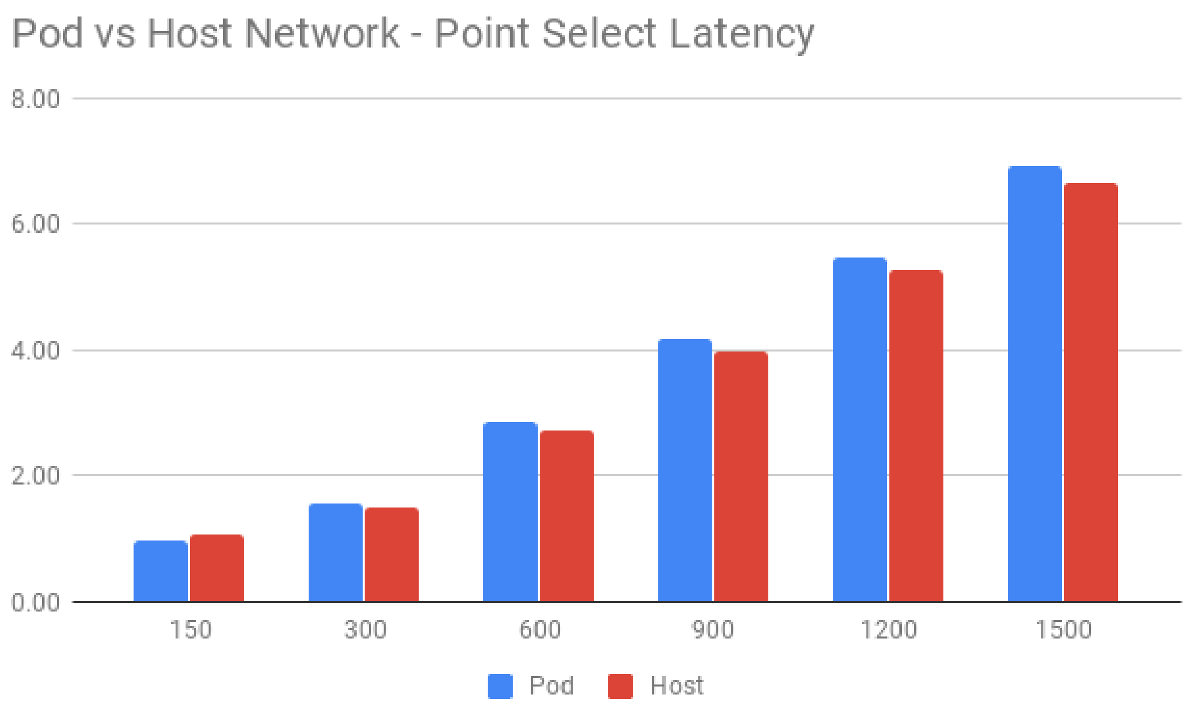


Figure 4: Pod vs Host Network

From the images above, the performance in Host network mode is slightly better than that in Pod network.

3.1.3.1.2 Ubuntu vs COS

GKE provides Ubuntu and COS for each node. In this test, the Point Select test of TiDB is conducted on both systems.

The network mode is 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 comparison:

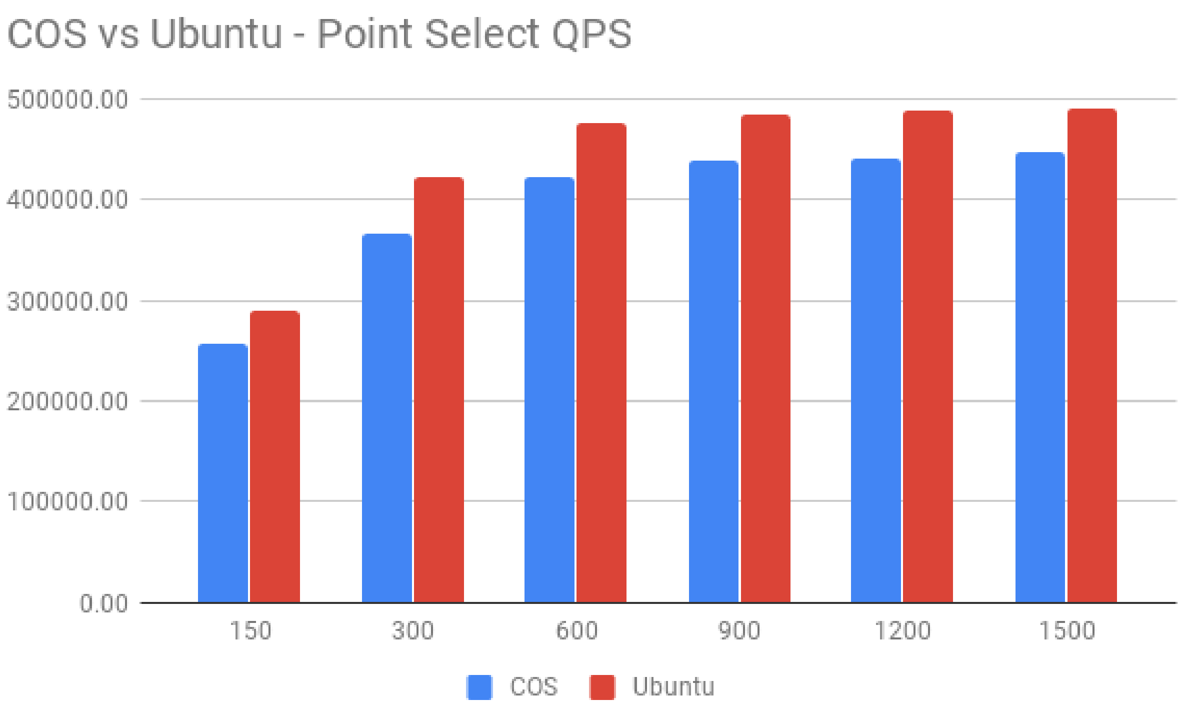


Figure 5: COS vs Ubuntu

Latency comparison:

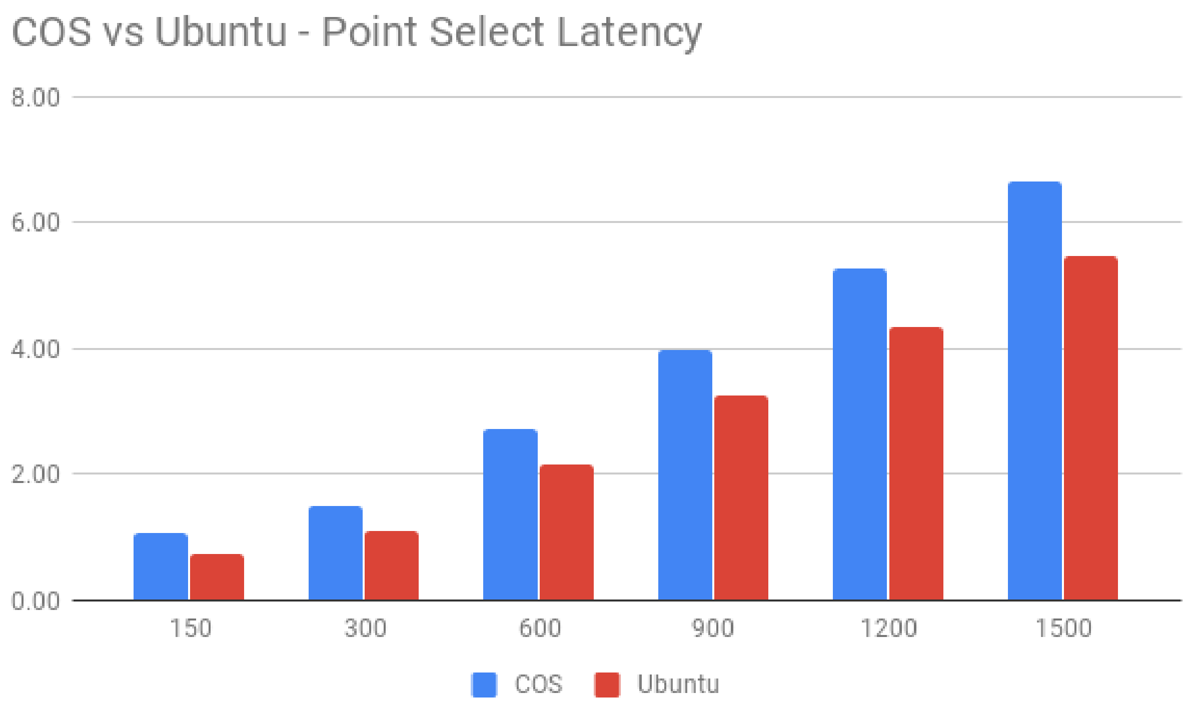


Figure 6: COS vs Ubuntu

From the images above, TiDB performs better on Ubuntu than on COS in the Point Select test.

Note:

- This test is conducted only for the single test case and indicates that the performance might be affected by different operating systems, different optimization and default settings. Therefore, PingCAP makes no recommendation for the operating system.
- COS is officially recommended by GKE, because it is optimized for containers and improved substantially on security and disk performance.

3.1.3.1.3 Kubernetes Service vs GCP LoadBalancer

After TiDB is deployed on Kubernetes, there are two ways of accessing TiDB: via Kubernetes Service inside the cluster, or via Load Balancer IP outside the cluster. TiDB is tested in both ways.

In this test, the operating system is Ubuntu and the network mode is Host.

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 comparison:

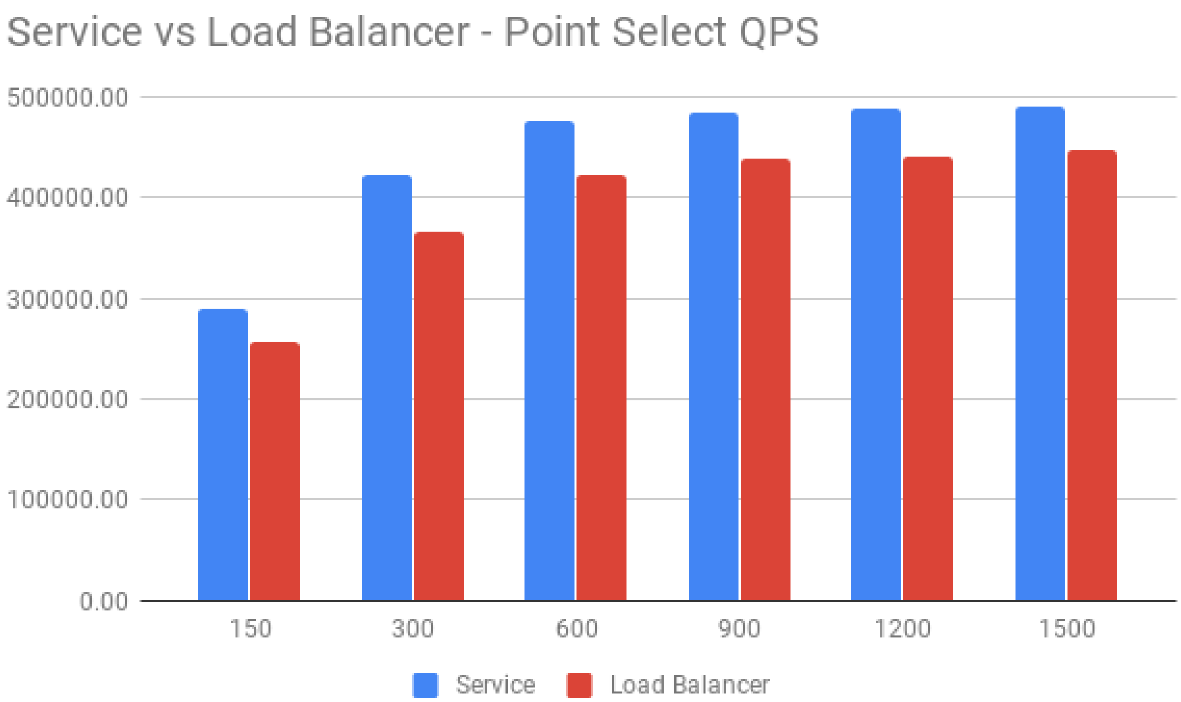


Figure 7: Service vs Load Balancer

Latency comparison:

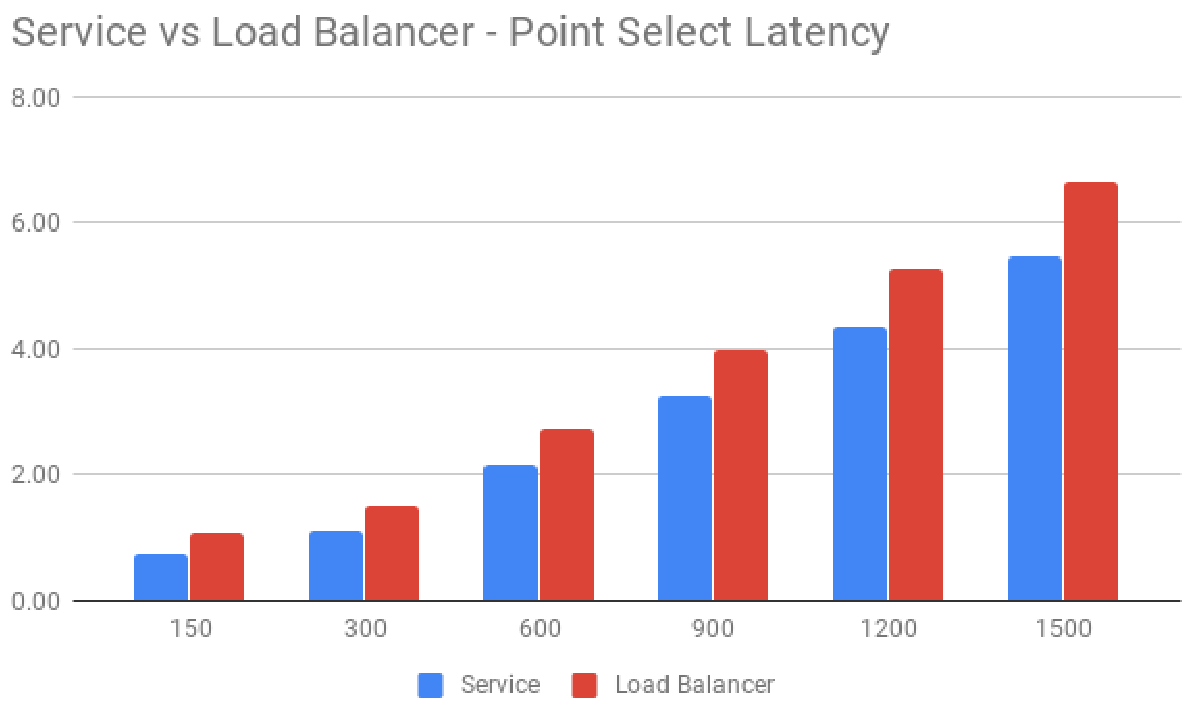


Figure 8: Service vs Load Balancer

From the images above, TiDB performs better when accessed via Kubernetes Service than accessed via GCP Load Balancer in the Point Select test.

3.1.3.1.4 n1-standard-16 vs c2-standard-16

In the Point Select read test, TiDB's CPU usage exceeds 1400% (16 cores) while TiKV's CPU usage is about 1000% (16 cores).

The test compares the TiDB performance on general machine types with that on machines which are optimized for computing. In this performance comparison, the frequency of n1-stadnard-16 is about 2.3G, and the frequency of c2-standard-16 is about 3.1G.

In this test, the operating system is Ubuntu and the Pod network is Host. TiDB is accessed via Kubernetes Service.

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

Threads	QPS	95% latency(ms)
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 comparison:

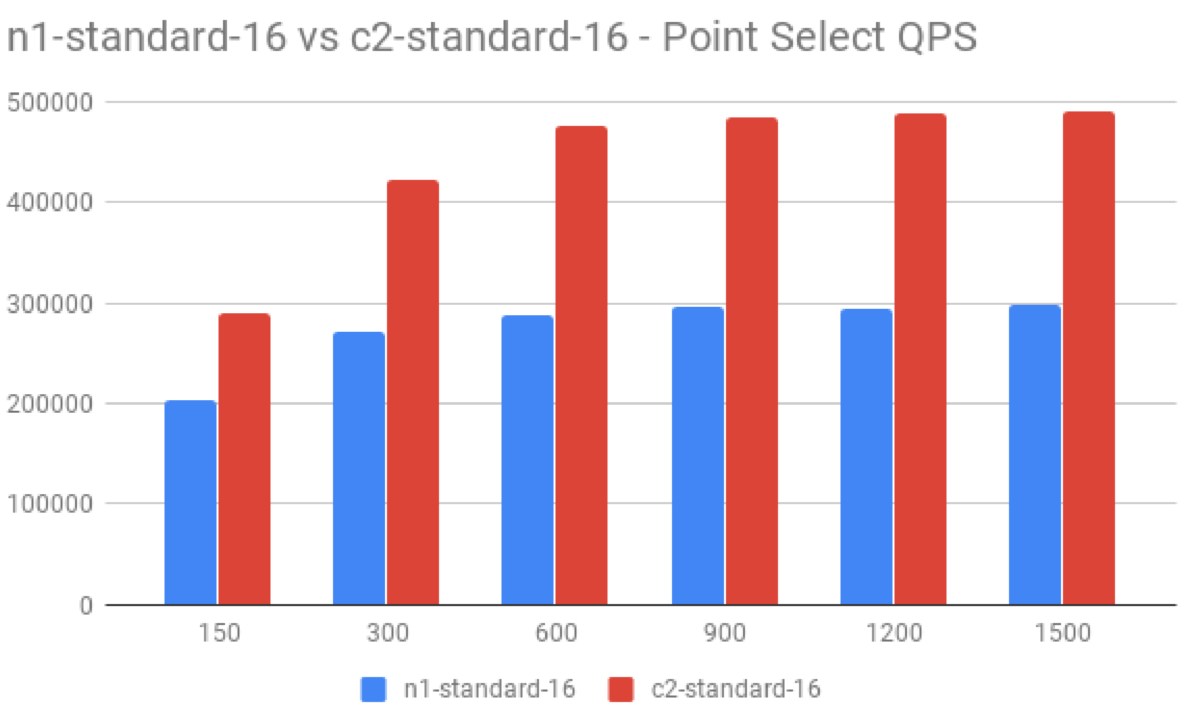


Figure 9: n1-standard-16 vs c2-standard-16

Latency comparison:

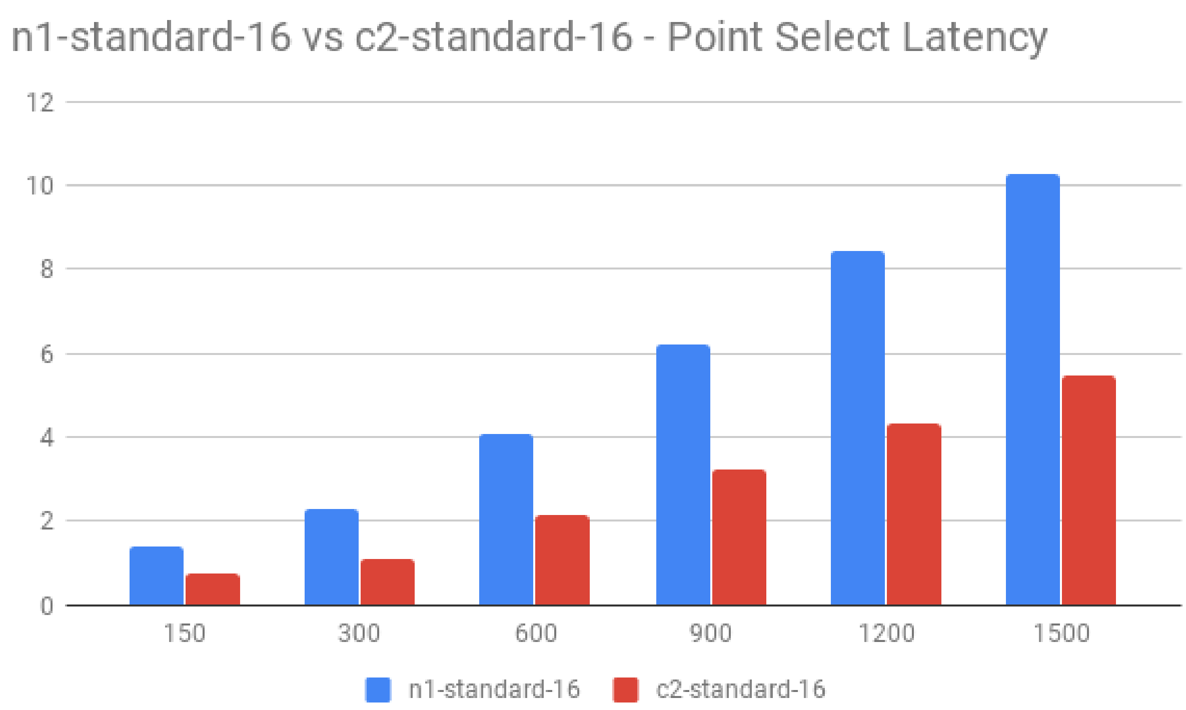


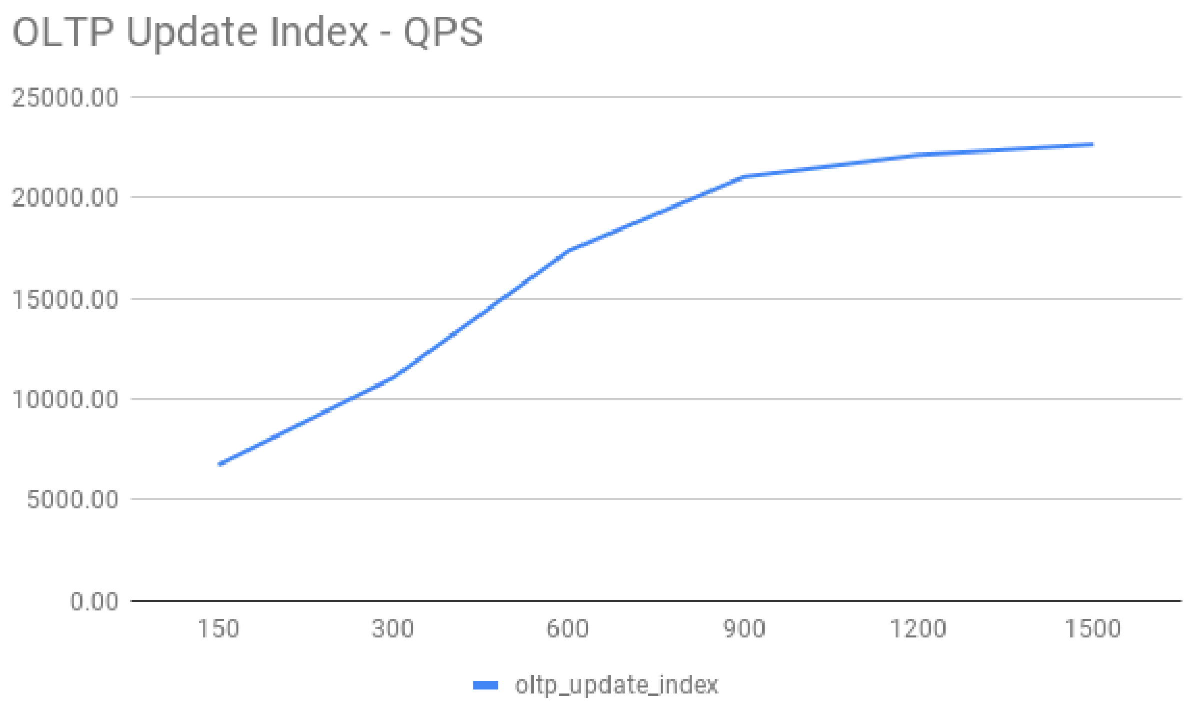
Figure 10: n1-standard-16 vs c2-standard-16

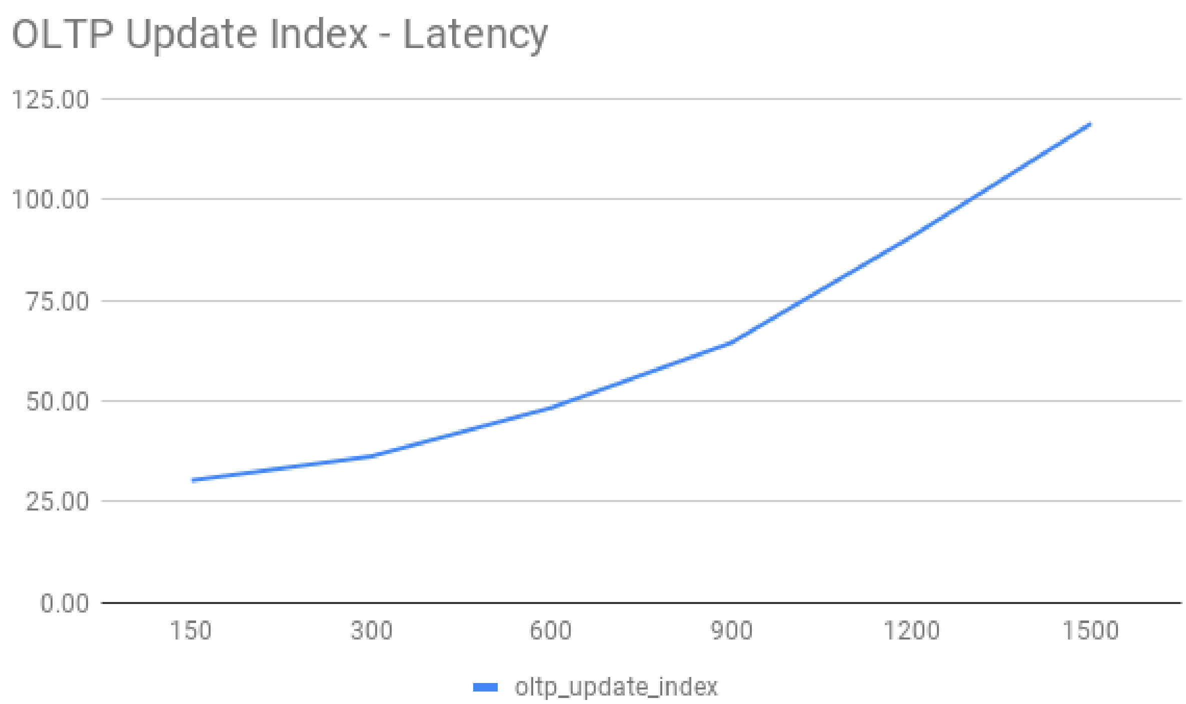
3.1.3.2 OLTP and other tests

The Point Select test is conducted on different operating systems and in different network modes, and the test results are compared. In addition, other tests in the OLTP test set are also conducted on Ubuntu in Host network mode where the TiDB cluster is accessed via Kubernetes Service.

3.1.3.2.1 OLTP Update Index

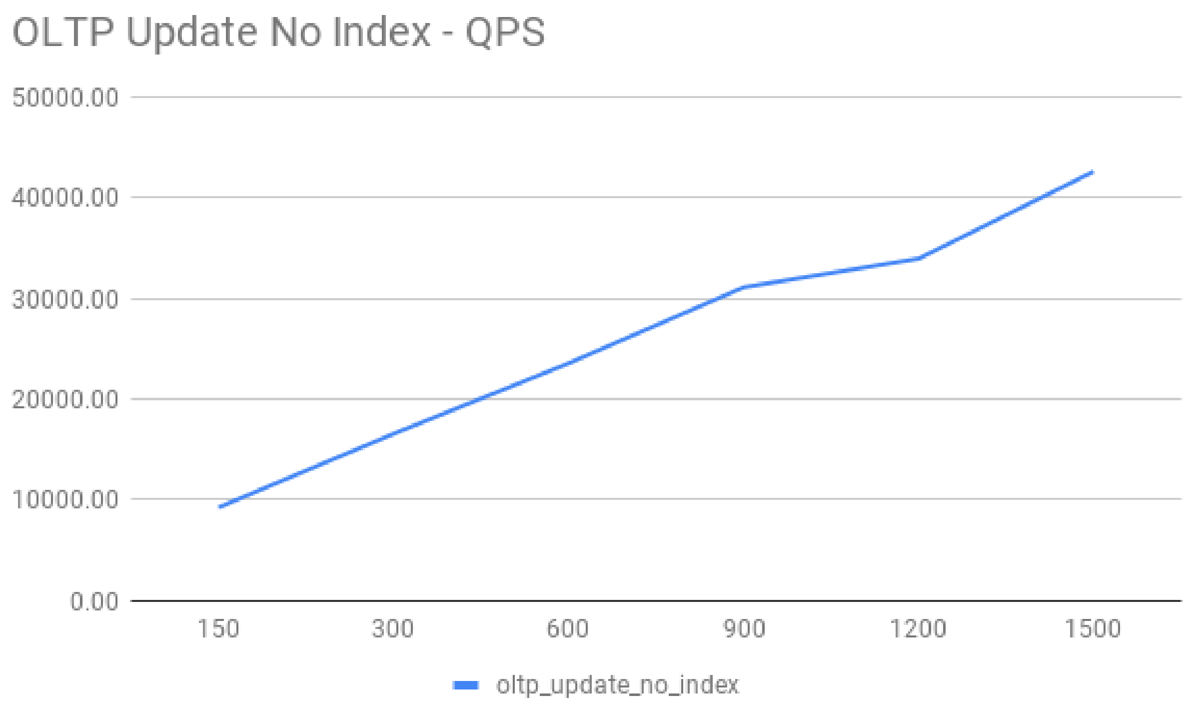
Threads	QPS	95% latency(ms)
150	6726.59	30.26
300	11067.55	36.24
600	17358.46	48.34
900	21025.23	64.47
1200	22121.87	90.78
1500	22650.13	118.92

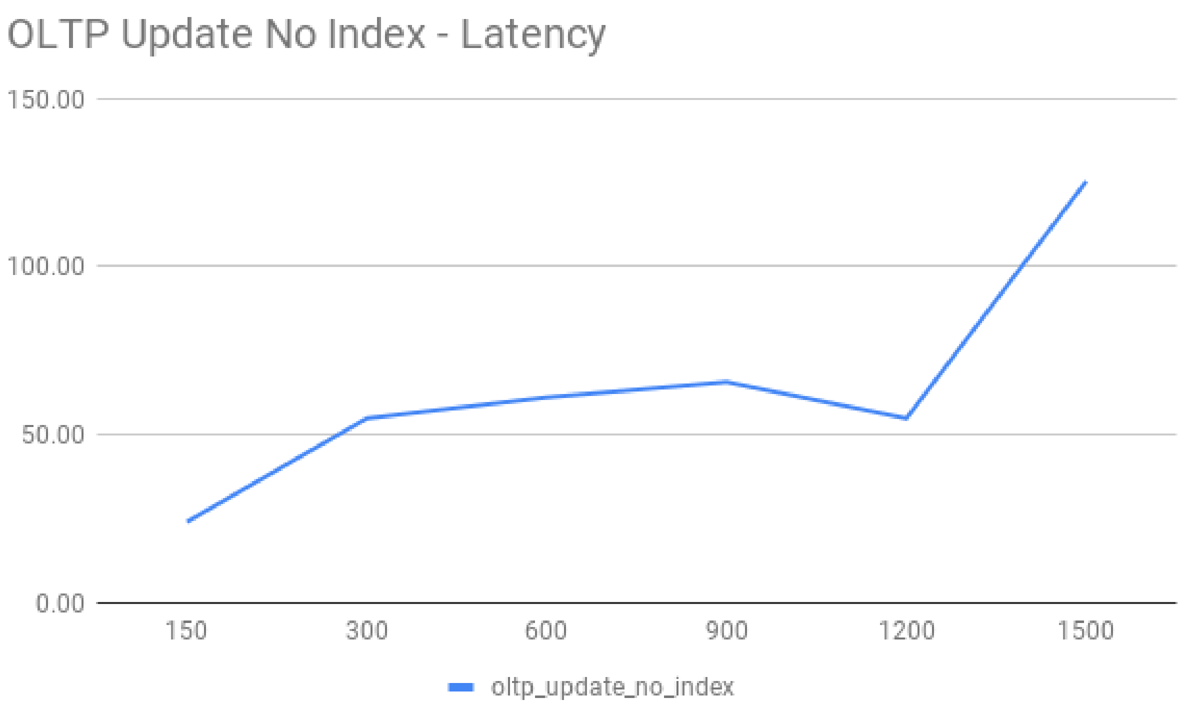




3.1.3.2.2 OLTP Update Non Index

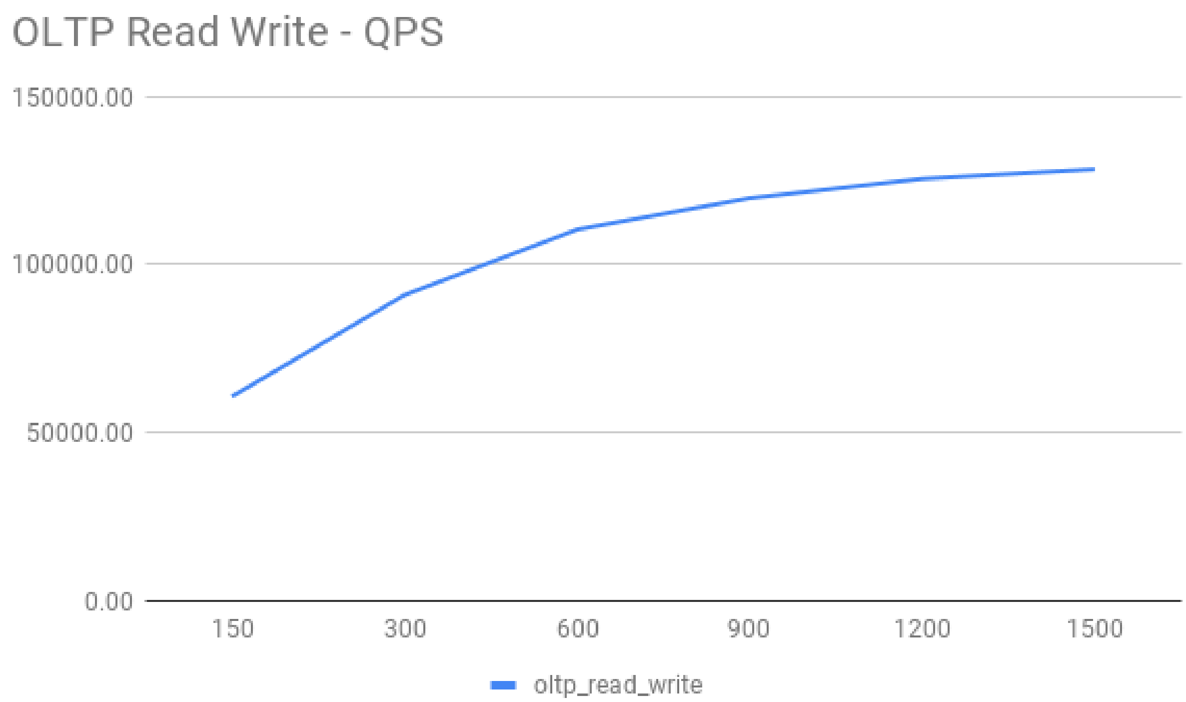
Threads	QPS	95% latency(ms)
150	9230.60	23.95
300	16543.63	54.83
600	23551.01	61.08
900	31100.10	65.65
1200	33942.60	54.83
1500	42603.13	125.52

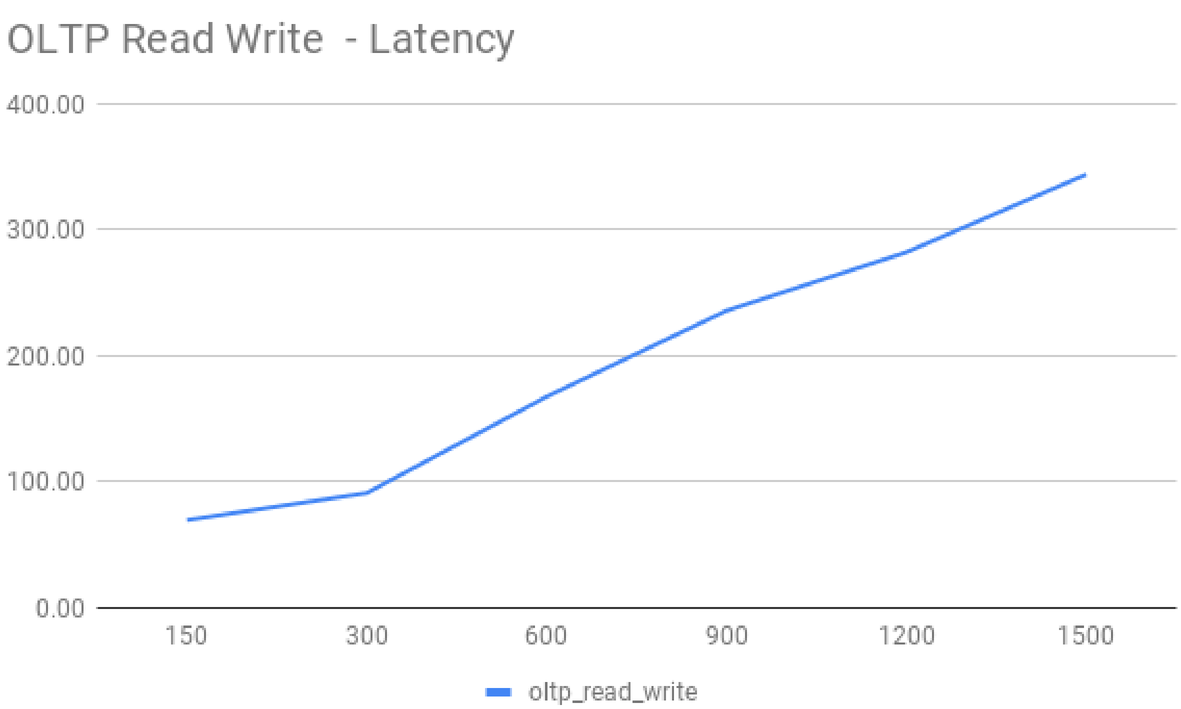




3.1.3.2.3 OLTP Read Write

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 Performance comparison between single AZ and multiple AZs

The network latency on communication across multiple AZs in GCP is slightly higher than that within the same zone. In this test, machines of the same configuration are used in different deployment plans under the same standard. The purpose is to learn how the latency across multiple AZs might affect the performance of TiDB.

Single AZ:

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

Multiple AZs:

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 comparison:

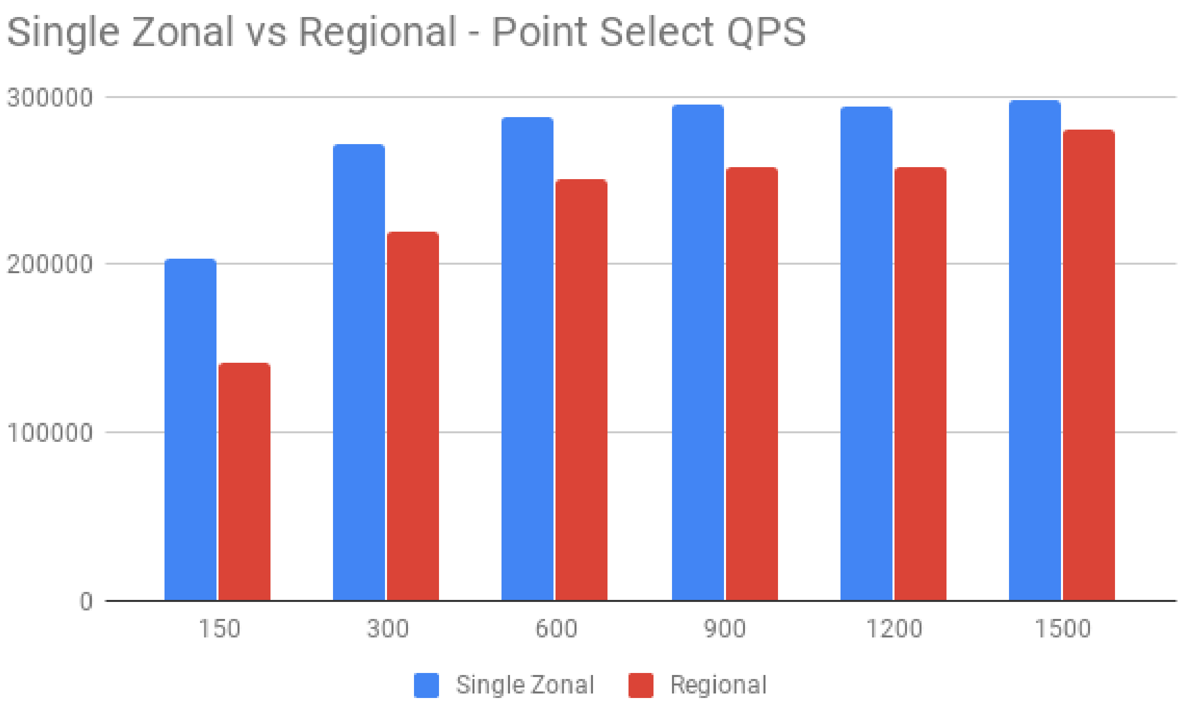


Figure 11: Single Zonal vs Regional

Latency comparison:

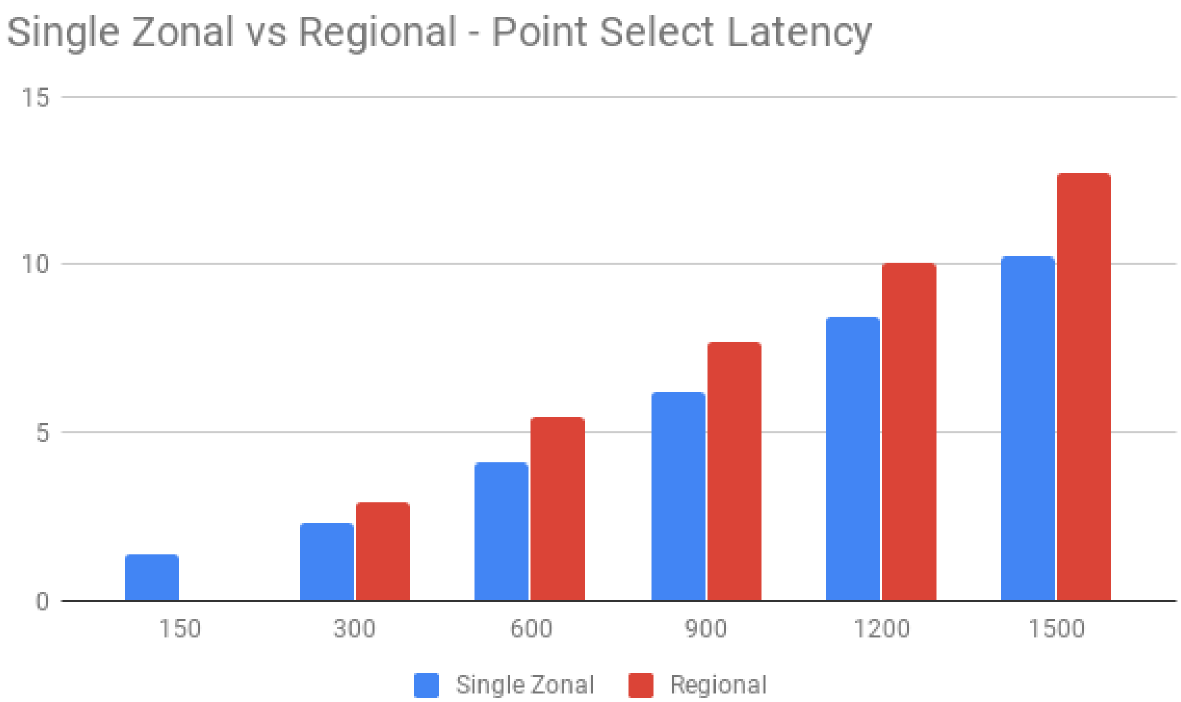


Figure 12: Single Zonal vs Regional

From the images above, the impact of network latency goes down as the concurrency pressure increases. In this situation, the extra network latency is no longer the main bottleneck of performance.

3.1.4 Conclusion

This is a test of TiDB using sysbench running in Kubernetes deployed on a typical public cloud platform. The purpose is to learn how different factors might affect the performance of TiDB. On the whole, these influencing factors include the following items:

- In the VPC-Native mode, TiDB performs slightly better in Host network than in Pod network. (The difference, ~7%, is measured in QPS. Performance differences caused by the factors below are also measured by QPS.)
- In Host network, TiDB performs better (~9%) in the read test on Ubuntu provided by GCP than on COS.
- The TiDB performance is slightly lower (~5%) if it is accessed outside the cluster via Load Balancer.
- Increased latency among nodes in multiple AZs has a certain impact on the TiDB performance (30% ~ 6%; the impact diminishes as the concurrent number increases).

- The QPS performance is greatly improved (50% ~ 60%) if the Point Select read test is conducted on machines of computing type (compared with general types), because the test mainly consumes CPU resources.

Note:

- The factors above might change over time. The TiDB performance might varies on different cloud platforms. In the future, more tests will be conducted on more dimensions.
- The sysbench test case cannot fully represent the actual business scenarios. It is recommended that you simulate the actual business for test and make consideration based on all the costs behind (machines, difference between operating systems, the limit of Host network, and so on).

4 Get Started

4.1 Deploy TiDB on Google Cloud

This tutorial is designed to be directly run in Google Cloud Shell.

It takes you through the following steps:

- Launch a new 3-node Kubernetes cluster (optional)
- Install the Helm package manager for Kubernetes
- Deploy the TiDB Operator
- Deploy your first TiDB cluster
- Connect to the TiDB cluster
- Scale out the TiDB cluster
- Shut down the Kubernetes cluster (optional)

Warning:

This is for testing only. DO NOT USE in production!

4.1.1 Select a project

This tutorial launches a 3-node Kubernetes cluster of `n1-standard-1` machines. Pricing information can be found here.

Please select a project before proceeding:

4.1.2 Enable API access

This tutorial requires use of the Compute and Container APIs. Please enable them before proceeding:

4.1.3 Configure gcloud defaults

This step defaults gcloud to your preferred project and zone, which simplifies the commands used for the rest of this tutorial:

```
gcloud config set project {{project-id}}
```

```
gcloud config set compute/zone us-west1-a
```

4.1.4 Launch a 3-node Kubernetes cluster

It's now time to launch a 3-node kubernetes cluster! The following command launches a 3-node cluster of `n1-standard-1` machines.

It takes a few minutes to complete:

```
gcloud container clusters create tidb
```

Once the cluster has launched, set it to be the default:

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

The last step is to verify that `kubectl` can connect to the cluster, and all three machines are running:

```
kubectl get nodes
```

If you see `Ready` for all nodes, congratulations! You've setup your first Kubernetes cluster.

4.1.5 Install Helm

Helm is the package manager for Kubernetes, and is what allows us to install all of the distributed components of TiDB in a single step. Helm requires both a server-side and a client-side component to be installed.

Install `helm`:

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

Copy helm to your $HOME directory so that it persists after the Cloud Shell reaches its idle timeout:

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

Helm also needs a couple of permissions to work properly:

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

It takes a minute for helm to initialize `tiller`, its server component:

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

When you see `Running`, it's time to hit Ctrl+C and proceed to the next step!

4.1.6 Add Helm repo

Helm repo (`https://charts.pingcap.org/`) houses PingCAP managed charts, such as tidb-operator, tidb-cluster and tidb-backup, etc. Add and check the repo with following commands:

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

Then you can check the available charts:

```
helm repo update
```

```
helm search tidb-cluster -l
```

```
helm search tidb-operator -l
```

4.1.7 Deploy TiDB Operator

Note that `<chartVersion>` is used in the rest of the document to represent the chart version, e.g. `v1.0.0`.

The first TiDB component we are going to install is the TiDB Operator, using a Helm Chart. TiDB Operator is the management system that works with Kubernetes to bootstrap your TiDB cluster and keep it running. This step assumes you are in the `tidb-operator` working directory:

```
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>
```

We can watch the operator come up with:

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

When you see both tidb-scheduler and tidb-controller-manager are `Running`, press Ctrl+C and proceed to launch a TiDB cluster!

4.1.8 Deploy your first TiDB cluster

Now with a single command we can bring-up a full TiDB cluster:

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

It takes a few minutes to launch. You can monitor the progress with:

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

The TiDB cluster includes 2 TiDB pods, 3 TiKV pods, and 3 PD pods. When you see all pods `Running`, it's time to Ctrl+C and proceed forward!

4.1.9 Connect to the TiDB cluster

There can be a small delay between the pod being up and running, and the service being available. You can watch list services available with:

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

When you see `demo-tidb` appear, you can Ctrl+C. The service is ready to connect to!

To connect to TiDB within the Kubernetes cluster, you can establish a tunnel between the TiDB service and your Cloud Shell. This is recommended only for debugging purposes, because the tunnel will not automatically be transferred if your Cloud Shell restarts. To establish a tunnel:

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

From your Cloud Shell:

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

Try out a MySQL command inside your MySQL terminal:

```sql
select tidb_version();
```

If you did not specify a password in helm, set one now:

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

> **Note:**
>
> This command contains some special characters which cannot be auto-populated in the google cloud shell tutorial, so you might need to copy and paste it into your console manually.

Congratulations, you are now up and running with a distributed TiDB database compatible with MySQL!

4.1.10 Scale out the TiDB cluster

With a single command we can easily scale out the TiDB cluster. To scale out TiKV:

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

Now the number of TiKV pods is increased from the default 3 to 5. You can check it with:

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

4.1.11 Accessing the Grafana dashboard

To access the Grafana dashboards, you can create a tunnel between the Grafana service and your shell. To do so, use the following command:

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

In Cloud Shell, click on the Web Preview button and enter 3000 for the port. This opens a new browser tab pointing to the Grafana dashboards. Alternatively, use the following URL https://ssh.cloud.google.com/devshell/proxy?port=3000 in a new tab or window.

If not using Cloud Shell, point a browser to `localhost:3000`.

4.1.12 Destroy the TiDB cluster

When the TiDB cluster is not needed, you can delete it with the following command:

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

The above commands only delete the running pods, the data is persistent. If you do not need the data anymore, you should run the following commands to clean the data and the dynamically created persistent disks:

```
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 Shut down the Kubernetes cluster

Once you have finished experimenting, you can delete the Kubernetes cluster with:

```
gcloud container clusters delete tidb
```

4.2 Deploy TiDB in the Minikube Cluster

This document describes how to deploy a TiDB cluster in the minikube cluster.

> **Warning:**
>
> This is for testing only. DO NOT USE in production!

4.2.1 Start a Kubernetes cluster with minikube

Minikube can start a local Kubernetes cluster inside a VM on your laptop. It works on macOS, Linux, and Windows.

> **Note:**
>
> Although Minikube supports `--vm-driver=none` that uses host docker instead of VM, it is not fully tested with TiDB Operator and may not work.

4.2.1.1 Install minikube and start a Kubernetes cluster

See Installing Minikube to install minikube (1.0.0+) on your machine.

After you installed minikube, you can run the following command to start a Kubernetes cluster.

```
minikube start
```

For Chinese mainland users, you may use local gcr.io mirrors such as `registry.cn-hangzhou.aliyuncs.com/google_containers`.

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

Or you can configure HTTP/HTTPS proxy environments in your Docker:

```
## change 127.0.0.1:1086 to your http/https proxy server IP:PORT
minikube start --docker-env https_proxy=http://127.0.0.1:1086 \
  --docker-env http_proxy=http://127.0.0.1:1086
```

Note:

As minikube is running with VMs (default), the `127.0.0.1` is the VM itself, you might want to use your real IP address of the host machine in some cases.

See minikube setup for more options to configure your virtual machine and Kubernetes cluster.

4.2.1.2 Install kubectl to access the cluster

The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters.

Install kubectl according to the instructions in Install and Set Up kubectl.

After kubectl is installed, test your minikube Kubernetes cluster:

```
kubectl cluster-info
```

4.2.2 Install TiDB Operator and run a TiDB cluster

4.2.2.1 Install helm

Helm is the package manager for Kubernetes and is what allows us to install all of the distributed components of TiDB in a single step. Helm requires both a server-side and a client-side component to be installed.

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

Install helm tiller:

```
helm init
```

If you have limited access to gcr.io, you can try a mirror. For example:

```
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]+')
```

Once it is installed, running `helm version` returns both the client and server version. For example:

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

If it shows only the client version, `helm` cannot yet connect to the server. Use `kubectl` to see if any tiller pods are running.

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

4.2.2.2 Add Helm repo

Helm repo (`https://charts.pingcap.org/`) houses PingCAP managed charts, such as tidb-operator, tidb-cluster and tidb-backup, etc. Add and check the repo with following commands:

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

Then you can check the available charts:

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

4.2.2.3 Install TiDB Operator in the Kubernetes cluster

> **Note:**
>
> `<chartVersion>` will be used in the rest of the document to represent the chart version, e.g. `v1.0.0`.

Clone tidb-operator repository:

```
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>
```

Now, you can watch the operator come up using the following command:

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

> **Note:**
>
> For Mac OS, if you are prompted "watch: command not found", you need to install the `watch` command using `brew install watch`. The same applies to other `watch` commands in this document.

If you have limited access to gcr.io (pods failed with ErrImagePull), you can try a mirror of kube-scheduler image. You can upgrade tidb-operator like this:

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

When you see both tidb-scheduler and tidb-controller-manager are running, you can process to launch a TiDB cluster!

4.2.2.4 Launch a TiDB cluster

To launch a TiDB cluster, use the following command:

```
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>
```

You can watch the cluster up and running using:

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

Use Ctrl+C to quit the watch mode.

4.2.2.5 Test a TiDB cluster

Before you start testing your TiDB cluster, make sure you have installed a MySQL client. Note that there can be a small delay between the time when the pod is up and running, and when the service is available. You can watch the list of available services with:

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

When you see `demo-tidb` appear, it's ready to connect to TiDB server.

To connect your MySQL client to the TiDB server, take the following steps:

1. Forward a local port to the TiDB port.

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

2. In another terminal window, connect the TiDB server with a MySQL client:

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

   Or you can run a SQL command directly:

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

4.2.2.6 Monitor TiDB cluster

To monitor the status of the TiDB cluster, take the following steps.

1. Forward a local port to the Grafana port.

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

2. Open your browser, and access Grafana at `http://localhost:3000`.

   Alternatively, Minikube provides `minikube service` that exposes Grafana as a service for you to access more conveniently.

   ```
   minikube service demo-grafana
   ```

   And it will automatically set up the proxy and open the browser for Grafana.

4.2.2.7 Delete TiDB cluster

Use the following commands to delete the demo cluster:

```
helm delete --purge demo

## update reclaim policy of PVs used by demo to Delete
kubectl get pv -l app.kubernetes.io/instance=demo -o name | xargs -I {}
    ↪ kubectl patch {} -p '{"spec":{"persistentVolumeReclaimPolicy":"Delete
    ↪ "}}'

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

4.2.3 FAQs

4.2.3.1 TiDB cluster in minikube is not responding or responds slow

The minikube VM is configured by default to only use 2048MB of memory and 2 CPUs. You can allocate more resources during `minikube start` using the `--memory` and `--cpus` flag. Note that you'll need to recreate minikube VM for this to take effect.

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

5 Deploy

5.1 Prerequisites for TiDB in Kubernetes

This document introduces the hardware and software prerequisites for deploying a TiDB cluster in Kubernetes.

5.1.1 Software version

Software Name	Version
Docker	Docker CE 18.09.6
Kubernetes	v1.12.5+
CentOS	7.6 and kernel 3.10.0-957 or later

5.1.2 The configuration of kernel parameters

Configuration Item	Value
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

When you set `net.bridge.bridge-nf-call-*` parameters, and if your option reports an error, you can check whether this module is loaded by running the following command:

```
lsmod|grep br_netfilter
```

If this module is not loaded, run the following command to load it:

```
modprobe br_netfilter
```

You also need to disable swap on each deployed Kubernetes node by running:

```
swapoff -a
```

To check whether swap is disabled:

```
free -m
```

If the above command shows that the swap column is all `0`, then swap is disabled.

In addition, to permanently disable swaps, remove all the swap-related entries in `/etc/` ↪ `fstab`.

After all above configurations are made, check whether you have configured SMP IRQ Affinity on the machine. This configuration is to assign the interrupt of each device to different CPUs to prevent all interrupts from being sent to the same CPU, avoiding potential performance bottleneck and taking advantage of multiple cores to increase cluster throughput. For the TiDB cluster, the rate at which the network card processes packages has a great impact on the throughput of the cluster.

Follow these steps to check whether you have configured SMP IRQ Affinity on the machine:

1. Execute the following command to check the interrupt of a network card:

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

    In the output result of the above command, the first column indicates the interrupt and the second column indicates the device name. If it is a multi-queue network card, the above command outputs information in multiple rows and each queue corresponds to an interrupt.

2. Execute either of the following commands to check this interrupt is assigned to which CPU.

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

    The above command outputs the hexadecimal value corresponding to the CPU serial number, and the output result is not so intuitive. For the detailed calculation method, refer to SMP IRQ Affinity.

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

    The above command outputs the decimal value corresponding to the CPU serial number. The result is more intuitive.

If all interrupts of a network card are assigned to different CPUs, the SMP IRQ Affinity is correctly configured on the machine and you do not need further operation.

If all interrupts are sent to the same CPU, configure SMP IRQ Affinity by the following steps:

- For the scenario of multi-queue network card and multiple cores:

    - Method 1: Enable the irqbalance service. Use the following command to enable the service on CentOS 7:

        ```
        systemctl start irqbalance
        ```

    - Method 2: Disable irqbalance and customize the binding relationship between interrupts and CPUs. Refer to the set_irq_affinity.sh script for more details.

- For the scenario of single-queue network card and multiple cores:

    To configure SMP IRQ Affinity in this scenario, you can use RPS/RFS to simulate the Receive Side Scaling (RSS) feature of the network card at the software level.

    Do not use the irqbalance service as described in Method 1. Instead, use the script provided in Method 2 to configure RPS. For the configuration of RFS, refer to RFS configuration.

5.1.3 Hardware and deployment requirements

- 64-bit generic hardware server platform in the Intel x86-64 architecture and 10 Gigabit NIC (network interface card), which are the same as the server requirements for deploying a TiDB cluster using binary. For details, refer to Hardware recommendations.

- The server's disk, memory and CPU choices depend on the capacity planning of the cluster and the deployment topology. It is recommended to deploy three master nodes, three etcd nodes, and several worker nodes to ensure high availability of the online Kubernetes cluster.

Meanwhile, the master node often acts as a worker node (that is, load can also be scheduled to the master node) to make full use of resources. You can set reserved resources by kubelet to ensure that the system processes on the machine and the core processes of Kubernetes have sufficient resources to run under high workloads. This ensures the stability of the entire system.

The following text analyzes the deployment plan of three Kubernetes masters, three etcd and several worker nodes. To achieve a highly available deployment of multi-master nodes in Kubernetes, see Kubernetes official documentation.

5.1.4 Kubernetes requirements for system resources

- It is required on each machine to have a relatively large SAS disk (at least 1T) to store the data directories of Docker and kubelet.

> **Note:**
> The data from Docker mainly includes image and container logs. The data from kubelet are mainly data used in emptyDir.

- If you need to deploy a monitoring system for the Kubernetes cluster and store the monitoring data on the disk, consider preparing a large SAS disk for Prometheus and also for the log monitoring system. This is also to guarantee that the purchased machines are homogeneous. For this reason, it is recommended to prepare two large SAS disks for each machine.

> **Note:**
> In a production environment, it is recommended to use RAID 5 for the two types of disks. You can decide how many disks for which you want to use RAID 5 as needed.

- It is recommended that the number of etcd nodes be consistent with that of the Kubernetes master nodes, and you store the etcd data on the SSD disk.

5.1.5 TiDB cluster's requirements for resources

The TiDB cluster consists of three components: PD, TiKV and TiDB. The following recommendations on capacity planning is based on a standard TiDB cluster, namely three PDs, three TiKVs and two TiDBs:

- PD component: 2C 4GB. PD occupies relatively less resources and only a small portion of local disks.

> **Note:**
> For easier management, you can put the PDs of all clusters on the master node. For example, to support five TiDB clusters, you can deploy five PD instances on each of the 3 master nodes. These PD instances use the same SSD disk (200 to 300 GigaBytes disk) on which you can create five directories as a mount point by means of bind mount. For detailed operation, refer to the documentation.
>
> If more machines are added to support more TiDB clusters, you can continue to add PD instances in this way on the master. If the resources on the master are exhausted, you can add PDs on other worker nodes in the same way. This method facilitates the planning and management of PD instances, while the downside is that if two machines go down, all TiDB clusters become unavailable due to the concentration of PD instances.
>
> Therefore, it is recommended to take out an SSD disk from each machine in all clusters and use it to provide PD instances like the master node. If you need to increase the capacity of a cluster by adding machines, you only need to create PD instances on the newly added machines.

- TiKV component: An NVMe disk of 8C 32GB for each TiKV instance. To deploy multiple TiKV instances on one machine, you must reserve enough buffers when planning capacity.
- TiDB component: 8C 32 GB capacity. Because the TiDB component does not occupy the disk space, you only need to consider the CPU and memory resources when planning. The following example assumes that the capacity is 8C 32 GB.

5.1.6 A case of planning TiDB clusters

This is an example of deploying five clusters (each cluster has 3 PDs, 3 TiKVs, and 2 TiDBs), where PD is configured as 2C 4GB, TiDB as 8C 32GB, and TiKV as 8C 32GB. There are seven Kubernetes nodes, three of which are both master and worker nodes, and the other four are purely worker nodes. The distribution of components on each node is as follows:

- Each master node:
  - 1 etcd (2C 4GB) + 2 PDs (2 * 2C 2 * 4GB) + 3 TiKVs (3 * 8C 3 * 32GB) + 1 TiDB (8C 32GB), totalling 38C 140GB
  - Two SSD disks, one for etcd and one for two PD instances
  - The RAID5-applied SAS disk used for Docker and kubelet
  - Three NVMe disks for TiKV instances
- Each worker node:
  - 3 PDs (3 * 2C 3 * 4GB) + 2 TiKVs (2 * 8C 2 * 32GB) + 2 TiDBs (2 * 8C 2 * 32GB), totalling 38C 140GB
  - One SSD disk for three PD instances
  - The RAID5-applied SAS disk used for Docker and kubelet
  - Two NVMe disks for TiKV instances

From the above analysis, a total of seven physical machines are required to support five sets of TiDB clusters. Three of the machines are master and worker nodes, and the remaining four are worker nodes. The configuration requirements for the machines are as follows:

- master and worker node: 48C 192GB, two SSD disks, one RAID5-applied SAS disk, three NVMe disks
- worker node: 48C 192GB, one block SSD disk, one RAID5-applied SAS disk, two NVMe disks

The above recommended configuration leaves plenty of available resources in addition to those taken by the components. If you want to add the monitoring and log components, use the same method to plan and purchase the type of machines with specific configurations.

Note:

In a production environment, avoid deploying TiDB instances on a master node due to the NIC bandwidth. If the NIC of the master node works at full capacity, the heartbeat report between the worker node and the master node will be affected and might lead to serious problems.

5.2 Deploy TiDB Operator in Kubernetes

This document describes how to deploy TiDB Operator in Kubernetes.

5.2.1 Prerequisites

Before deploying TiDB Operator, make sure the following items are installed on your machine:

- Kubernetes >= v1.12
- DNS addons
- PersistentVolume
- RBAC enabled (optional)
- Helm version >= 2.11.0 and < 2.16.4

> **Note:**
>
> - Although TiDB Operator can use network volume to persist TiDB data, this can affect performance a lot because TiDB stores multiple replicas itself. So it is highly recommended to set up local volume for better performance.
> - Network volumes in a multi-availability zone setup require Kubernetes v1.12 or higher version. It is recommended to use networked volumes to store backup data in `tidb-bakup` chart.

5.2.2 Deploy Kubernetes cluster

TiDB Operator runs in Kubernetes cluster. You can refer to the document of how to set up Kubernetes to set up a Kubernetes cluster. Make sure that the Kubernetes version is v1.12 or higher. If you are using AWS, GKE or local machines, here are quick-start tutorials:

- Google GKE tutorial
- AWS EKS tutorial

If you are deploying in a different environment, a proper DNS addon must be installed in the Kubernetes cluster. You can follow the official documentation to set up a DNS addon.

TiDB Operator uses Persistent Volume to persist the data of TiDB cluster (including the database, monitoring data, backup data), so the Kubernetes cluster must provide at least one kind of persistent volume. For better performance, it is recommended to use local SSD disk as the volumes. Follow this step to auto-provision local persistent volumes.

It is suggested to enable RBAC in the Kubernetes cluster. Otherwise, you need to set `rbac.create` to `false` in the `values.yaml` of both `tidb-operator` and `tidb-cluster` charts.

Because TiDB uses many file descriptors by default, the worker node and its Docker daemon's `ulimit` values must be greater than or equal to `1048576`.

1. Configure the `ulimit` value of the work node. See How to set `ulimit` values.

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

   Set the `nofile` values of `soft` and `hard` of the root account to be greater than or equal to `1048576`.

2. Configure the `ulimit` value of the Docker service.

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

   Set `LimitNOFILE` to be greater than or equal to `1048576`.

> **Note:**
> You need to explicitly set `LimitNOFILE` to `1048576` or a larger value rather than the default `infinity`. Because of a bug in `systemd`, the value of `infinity` is `65536` in some `systemd` versions.

5.2.3 Install Helm

Refer to Use Helm to install Helm and configre it with the official PingCAP chart Repo.

5.2.4 Configure local persistent volume

5.2.4.1 Prepare local volumes

Refer to Local PV Configuration to set up local persistent volumes in your Kubernetes cluster.

5.2.5 Install TiDB Operator

TiDB Operator uses Custom Resource Definition (CRD) to extend Kubernetes. Therefore, to use TiDB Operator, you must first create the `TidbCluster` custom resource type, which is a one-time job in your Kubernetes cluster.

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

After `TidbCluster` custom resource type is created, install TiDB Operator in your Kubernetes cluster.

1. Get the `values.yaml` file of the `tidb-operator` chart you want to install.

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

> **Note:**
> `<chart-version>` represents the chart version of TiDB Operator. For example, `v1.0.0`. You can view the currently supported versions by running the `helm search -l tidb-operator` command.

2. Install 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 Customize TiDB Operator

To customize TiDB Operator, modify `/home/tidb/tidb-operator/values-tidb-↪ operator.yaml`. The rest sections of the document use `values.yaml` to refer to `/home/tidb/tidb-operator/values-tidb-operator.yaml`

TiDB Operator contains two components:

- tidb-controller-manager
- tidb-scheduler

These two components are stateless and deployed via `Deployment`. You can customize resource `limit`, `request`, and `replicas` in the `values.yaml` file.

After modifying `values.yaml`, run the following command to apply this modification:

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

5.3 Deploy TiDB on General Kubernetes

This document describes how to deploy a TiDB cluster on general Kubernetes.

5.3.1 Prerequisites

- Deploy TiDB Operator;
- Install Helm and configure it with the official PingCAP chart.

5.3.2 Configure TiDB cluster

Use the following commands to get the `values.yaml` configuration file of the tidb-cluster chart to be deployed.

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

Note:

- You can replace `/home/tidb` with any directory as you like.
- `release-name` is the prefix of resources used by TiDB in Kubernetes (such as Pod, Service, etc.). You can give it a name that is easy to memorize but this name must be *globally unique*. You can view existing `release-names` in the cluster by running the `helm ls -q` command.
- `chart-version` is the version released by the `tidb-cluster` chart. You can view the currently supported versions by running the `helm search ↪ -l tidb-cluster` command.
- In the rest of this document, `values.yaml` refers to `/home/tidb/< ↪ release-name>/values-<releaseName>.yaml`.

5.3.2.1 Storage class

The TiDB cluster uses `local-storage` by default.

- For the production environment, local storage is recommended. The actual local storage in Kubernetes clusters might be sorted by disk types, such as `nvme-disks` and `sas-disks`.
- For the demonstration environment or functional verification, you can use network storage, such as `ebs` and `nfs`.

Different components of a TiDB cluster have different disk requirements. Before deploying a TiDB cluster, select the appropriate storage class for each component according to the storage classes supported by the current Kubernetes cluster and usage scenario. You can set the storage class by modifying `storageClassName` of each component in `values.yaml ↪` . For the storage classes supported by the Kubernetes cluster, check with your system administrator.

Note:

If you set a storage class that does not exist in the TiDB cluster that you are creating, then the cluster creation goes to the Pending state. In this situation, you must destroy the TiDB cluster in Kubernetes.

5.3.2.2 Cluster topology

The deployed cluster topology by default has 3 PD Pods, 3 TiKV Pods, 2 TiDB Pods, and 1 Monitor Pod. In this deployment topology, the scheduler extender of TiDB Operator requires at least 3 nodes in the Kubernetes cluster to provide high availability. If the number of Kubernetes cluster nodes is less than 3, 1 PD Pod goes to the Pending state, and neither TiKV Pods nor TiDB Pods are created.

When the number of nodes in the Kubernetes cluster is less than 3, to start the TiDB cluster, you can reduce both the number of PD Pods and TiKV Pods in the default deployment to `1`, or modify the `schedulerName` in `values.yaml` to `default-scheduler`, a built-in scheduler in Kubernetes.

Warning:

`default-scheduler` is only applicable to the demonstration environment. After `schedulerName` is modified to `default-scheduler`, the scheduling of TiDB clusters neither guarantees high availability of data nor supports features such as TiDB stable scheduling.

For more configuration parameters, see TiDB cluster configurations in Kubernetes.

5.3.3 Deploy TiDB Cluster

After you deploy and configure TiDB Operator, deploy the TiDB cluster using the following commands:

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

Note:

A `namespace` is a virtual cluster backed by the same physical cluster. You can give it a name that is easy to memorize, such as the same name as `release-name`.

You can view the Pod status using the following command:

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

You can use TiDB Operator to deploy and manage multiple sets of TiDB clusters in a single Kubernetes cluster by repeating the above command and replacing `release-name` with a different name. Different clusters can be in the same or different `namespace`. You can select different clusters according to your actual needs.

5.4 Deploy TiDB on AWS EKS

This document describes how to deploy a TiDB cluster on AWS EKS with your laptop (Linux or macOS) for development or testing.

5.4.1 Prerequisites

Before deploying a TiDB cluster on AWS EKS, make sure the following requirements are satisfied:

- awscli >= 1.16.73, to control AWS resources

  You must configure `awscli` before it can interact with AWS. The fastest way is using the `aws configure` command:

  ```
  aws configure
  ```

  Replace AWS Access Key ID and AWS Secret Access Key with your own keys:

  ```
  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
  ```

Note:

The access key must have at least permissions to: create VPC, create EBS, create EC2 and create role.

- terraform >= 0.12
- kubectl >= 1.12
- helm >= 2.11.0 and < 2.16.4
- jq
- aws-iam-authenticator installed in `PATH`, to authenticate with AWS

  The easiest way to install `aws-iam-authenticator` is to download the prebuilt binary as shown below:

  Download the binary for Linux:

  ```
  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
  ```

  Or, download binary for macOS:

  ```
  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
  ```

  Then execute the following commands:

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

5.4.2 Deploy

The default setup creates a new VPC and a `t2.micro` instance as the bastion machine, and an EKS cluster with following Amazon EC2 instances as worker nodes:

- 3 m5.xlarge instances for PD
- 3 c5d.4xlarge instances for TiKV
- 2 c5.4xlarge instances for TiDB
- 1 c5.2xlarge instance for monitor

Use the following commands to set up the cluster.

Get the code from Github:

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

Apply the configs, note that you must answer "yes" to `terraform apply` to continue:

```
terraform init
```

```
terraform apply
```

It might take 10 minutes or more to finish the process. After `terraform apply` is executed successfully, some useful information is printed to the console.

A successful deployment will give the output like:

```
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-2
```

You can use the `terraform output` command to get the output again.

Note:

EKS versions earlier than 1.14 do not support auto enabling cross-zone load balancing via Network Load Balancer (NLB). Therefore, unbalanced pressure distributed among TiDB instances can be expected in default settings. It is strongly recommended that you refer to AWS Documentation to manually enable cross-zone load balancing for a production environment.

5.4.3 Access the database

To access the deployed TiDB cluster, use the following commands to first `ssh` into the bastion machine, and then connect it via MySQL client (replace the `<>` parts with values from the output):

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

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

The default value of `eks_name` is `my-cluster`. If the DNS name is not resolvable, be patient and wait a few minutes.

You can interact with the EKS cluster using `kubectl` and `helm` with the kubeconfig file `credentials/kubeconfig_<eks_name>` in the following two ways.

- By specifying `--kubeconfig` argument:

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

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

- Or by setting the `KUBECONFIG` environment variable:

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

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

```
helm ls
```

5.4.4 Monitor

You can access the `<monitor-dns>:3000` address (printed in outputs) using your web browser to view monitoring metrics.

The initial Grafana login credentials are:

- User: admin
- Password: admin

5.4.5 Upgrade

To upgrade the TiDB cluster, edit the `variables.tf` file with your preferred text editor and modify the `default_cluster_version` variable to a higher version, and then run `terraform apply`.

For example, to upgrade the cluster to version 3.0.1, modify the `default_cluster_version` ↪ to `v3.0.1`:

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

Note:

The upgrading doesn't finish immediately. You can watch the upgrading process by `kubectl --kubeconfig credentials/kubeconfig_<eks_name> ↪ get po -n <default_cluster_name> --watch`.

5.4.6 Scale

To scale the TiDB cluster, edit the `variables.tf` file with your preferred text editor and modify the `default_cluster_tikv_count` or `default_cluster_tidb_count` variable to your desired count, and then run `terraform apply`.

For example, to scale out the cluster, you can modify the number of TiDB instances from 2 to 4:

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

Note:

Currently, scaling in is NOT supported because we cannot determine which node to scale. Scaling out needs a few minutes to complete, you can watch the scaling out by `kubectl --kubeconfig credentials/kubeconfig_< ↪ eks_name> get po -n <default_cluster_name> --watch`.

5.4.7 Customize

You can change default values in `variables.tf` (such as the cluster name and image versions) as needed.

5.4.7.1 Customize AWS related resources

By default, the terraform script will create a new VPC. You can use an existing VPC by setting `create_vpc` to `false` and specify your existing VPC id and subnet ids to `vpc_id`, `private_subnet_ids` and `public_subnet_ids` variables.

Note:

- Reusing VPC and subnets of an existing EKS cluster is not supported yet due to limitations of AWS and Terraform, so only change this option if you have to use a manually created VPC.
- The CNI plug-in on the EKS Node reserves some IP resources for each node. When manually creating a VPC, it is recommended to set the subnet mask length to 18~20 to ensure sufficient IP resources, or configure the CNI plugin to reserver less IP resources according to EKS CNI plugin documentation.

An Amazon EC2 instance is also created by default as the bastion machine to connect to the created TiDB cluster. This is because the TiDB service is exposed as an Internal Elastic Load Balancer. The EC2 instance has MySQL and Sysbench pre-installed, so you can use SSH to log into the EC2 instance and connect to TiDB using the ELB endpoint. You can disable the bastion instance creation by setting `create_bastion` to `false` if you already have an EC2 instance in the VPC.

The TiDB version and the number of components are also configurable in `variables.tf`. You can customize these variables to suit your needs.

Currently, the instance type of the TiDB cluster component is not configurable because PD and TiKV depend on NVMe SSD instance store, and different instance types have different disks.

5.4.7.2 Customize a TiDB cluster

The terraform scripts provide proper default settings for the TiDB cluster in EKS. You can specify an overriding values file - `values.yaml` through the `override_values` parameter in `clusters.tf` for each TiDB cluster. Values of this file will override the default settings.

For example, the default cluster uses `./default-cluster.yaml` as the overriding values file, and the ConfigMap rollout feature is enabled in this file.

In EKS, some configuration items are not customizable in `values.yaml`, such as the cluster version, replicas, `NodeSelector` and `Tolerations`. `NodeSelector` and `Tolerations` ↪ are controlled by Terraform to ensure consistency between the infrastructure and TiDB clusters. Cluster version and replicas can be modified in each `tidb-cluster` module in the `clusters.tf` file directly.

Note:

It is not recommended to include the following configurations (default configurations of `tidb-cluster` module) in the customized `values.yaml`:

```
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 Customize TiDB Operator

You can customize the TiDB Operator by specifying a Helm values file through the `operator_values` variable in the `variables.tf` file. For example:

```
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 Manage multiple TiDB clusters

An instance of `tidb-cluster` module corresponds to a TiDB cluster in the EKS cluster. If you want to add a new TiDB cluster, you can edit `./cluster.tf` and add a new instance of `tidb-cluster` module:

```
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
}
```

Note:

The `cluster_name` of each cluster must be unique.

You can get the addresses for TiDB and the monitoring service of the created cluster via `kubectl`. If you want the Terraform script to print this information, you can add `output` sections in `outputs.tf`:

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

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

When you finish modification, you can execute `terraform init` and `terraform apply` to create the TiDB cluster.

To delete the TiDB cluster, you can remove the `tidb-cluster` module in `cluster.tf`, execute `terraform apply` and the corresponding EC2 resources will be released as well.

5.4.9 Manage the infrastructure only

To configure the Terraform script to create only the Kubernetes cluster and TiDB Operator, take the following step:

Modify the `create_tidb_cluster_release` configuration item of the TiDB cluster in `clusters.tf`:

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

If `create_tidb_cluster_release` is set to `false`, the Terraform script does not create or modify the TiDB cluster. However, it still creates the computing and storage resources needed by the TiDB cluster. You can manage the cluster independently using tools like Helm.

Note:

If you set `create_tidb_cluster_release` to `false` in a cluster that has been deployed, the installed TiDB cluster will be deleted, and the corresponding TiDB cluster object will also be deleted.

5.4.10 Destroy clusters

It may take some time to finish destroying the cluster.

```
terraform destroy
```

Note:

- This will destroy your EKS cluster along with all the TiDB clusters you deployed on it.
- If you do not need the data on the volumes anymore, you have to manually delete the EBS volumes in AWS console after running `terraform ↪ destroy`.

5.4.11 Manage multiple Kubernetes clusters

This section describes the best practice to manage multiple Kubernetes clusters, each with one or more TiDB clusters installed.

The Terraform module in our case typically combines several sub-modules:

- `tidb-operator`, that provisions the Kubernetes control plane for TiDB cluster
- `tidb-cluster`, that creates the resource pool in the target Kubernetes cluster and deploy the TiDB cluster
- A `VPC` module, a `bastion` module and a `key-pair` module that are dedicated to TiDB on AWS

The best practice for managing multiple Kubernetes clusters is creating a new directory for each of your Kubernetes clusters, and combine the above modules according to your needs via Terraform scripts, so that the Terraform states among clusters do not interfere with each other, and it is convenient to expand. Here's an example:

```
## assume we are in the project root
mkdir -p deploy/aws-staging
vim deploy/aws-staging/main.tf
```

The content of `deploy/aws-staging/main.tf` could be:

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

```

```
## Creates an SSH key to log in the bastion and the Kubernetes node
module "key-pair" {
  source = "../modules/aws/key-pair"

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

## Provisions a VPC
module "vpc" {
  source = "../modules/aws/vpc"

  vpc_name = "another-eks-cluster"
}

## Provisions an EKS control plane with TiDB Operator installed
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
}

## HACK: enforces Helm to depend on the EKS
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
  }
}

## Provisions a TiDB cluster in the EKS cluster
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
}

## Provisions another TiDB cluster in the EKS cluster
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
}

## Provisions a bastion machine to access the TiDB service and worker nodes
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
}

## Prints the TiDB hostname of tidb-cluster-a
output "cluster-a_tidb-dns" {
  description = "tidb service endpoints"
  value      = module.tidb-cluster-a.tidb_hostname
}

## print the monitor hostname of tidb-cluster-b
output "cluster-b_monitor-dns" {
  description = "tidb service endpoint"
  value      = module.tidb-cluster-b.monitor_hostname
```

```
}

output "bastion_ip" {
  description = "Bastion IP address"
  value       = module.bastion.bastion_ip
}
```

As shown in the code above, you can omit most of the parameters in each of the module calls because there are reasonable defaults, and it is easy to customize the configuration. For example, just delete the bastion module call if you do not need it.

To customize each field, you can refer to the default Terraform module. Also, you can always refer to the `variables.tf` file of each module to learn about all the available parameters.

In addition, you can easily integrate these modules into your own Terraform workflow. If you are familiar with Terraform, this is our recommended way of use.

Note:

- When creating a new directory, please pay attention to its relative path to Terraform modules, which affects the `source` parameter during module calls.
- If you want to use these modules outside the tidb-operator project, make sure you copy the whole `modules` directory and keep the relative path of each module inside the directory unchanged.
- Due to limitation hashicorp/terraform#2430 of Terraform, the hack processing of Helm provider is necessary in the above example. It is recommended that you keep it in your own Terraform scripts.

If you are unwilling to write Terraform code, you can also copy the `deploy/aws` directory to create new Kubernetes clusters. But note that you cannot copy a directory that you have already run `terraform apply` against, when the Terraform state already exists in local. In this case, it is recommended to clone a new repository before copying the directory.

5.5 Deploy TiDB on GCP GKE

This document describes how to deploy a TiDB cluster on GCP GKE with your laptop (Linux or macOS) for development or testing.

Warning:

The GKE support for multiple disks per node has known issues that make it not ready for production usage. We are working to get GKE to resolve this issue.

5.5.1 Prerequisites

First of all, make sure the following items are installed on your machine:

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

5.5.2 Configure

To guarantee a smooth deployment, you need to do some configuration. Before configuring Google Cloud SDK, API, and Terraform, download the following resource:

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

5.5.2.1 Configure Cloud SDK

After installing Google Cloud SDK, run `gcloud init` to perform initial setup tasks.

5.5.2.2 Configure APIs

If the GCP project that you use is a new one, make sure the following APIs are enabled:

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

5.5.2.3 Configure Terraform

To execute the Terraform script, you need to configure the following three variables. You can configure them as prompted by Terraform, or define them in a `.tfvars` file.

- GCP_CREDENTIALS_PATH: Path to a valid GCP credentials file.
  - It is recommended for you to create a separate service account to be used by Terraform. See Creating and managing service accounts for more information. ./create-service-account.sh will create such a service account with minimal permissions.
  - See Creating and managing service account keys for information on creating service account keys. The steps in the script below detail how to do this using a script provided in the deploy/gcp directory, alternatively if creating the service account and key yourself, choose JSON key type during creation. The downloaded JSON file that contains the private key is the credentials file you need.
- GCP_REGION: The region in which to create the resources, for example: us-west1.
- GCP_PROJECT: The GCP project in which everything will be created.

To configure Terraform with the three variables above, perform the following steps:

1. Replace the GCP_REGION with your GCP region.

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

2. Replace the GCP_PROJECT with your GCP project name. Make sure you are connected to the correct project.

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

3. Initialize Terraform.

```
terraform init
```

4. Create a service account for Terraform with restricted permissions and set the credentials path.

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

Terraform automatically loads and populates variables from the files matching terraform.tfvars or *.auto.tfvars. For more information, see the Terraform documentation. The steps above will populate terraform.tfvars with GCP_REGION and GCP_PROJECT, and credentials.auto.tfvars with GCP_CREDENTIALS_PATH.

5.5.3 Deploy a TiDB cluster

This section describes how to deploy a TiDB cluster.

1. Decide on instance types.

   - If you just want to get a feel for a TiDB deployment and lower your cost, use the small settings:

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

   - If you want to benchmark a production deployment, use the production settings:

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

     The `prod.tfvars` setup creates a new VPC, two subnetworks, and an f1-micro instance as a bastion machine. This setup is created with the following instance types as worker nodes:

     - 3 n1-standard-4 instances for PD
     - 3 n1-highmem-8 instances for TiKV
     - 3 n1-standard-16 instances for TiDB
     - 3 n1-standard-2 instances for monitor

     The production setup, as listed above, requires at least 91 CPUs which exceed the default CPU quota of a GCP project. To increase your project's quota, follow these instructions. You need more CPUs if you need to scale out.

> **Note:**
> The number of worker nodes created depends on the number of Availability Zones in the specified region. Most regions have 3 zones, but `us-central1` has 4 zones. See Regions and zones for more information and see the Customize section on how to customize node pools in a regional cluster.

2. Execute the script to deploy the TiDB cluster.

   ```
   terraform apply
   ```

> **Note:**
> If you have not set the three variables above ahead of time, you might be prompted to set them when you run `terraform apply`. See Configure Terraform for details.

It might take 10 minutes or more to finish the process. A successful deployment gives the output like:

```
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 Access the TiDB database

After `terraform apply` is successful executed, perform the following steps to access the TiDB cluster. Replace the `<>` section with the output of running `terraform apply` above.

1. Connect to the bastion machine by using `ssh`.

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

2. Access the TiDB cluster via a MySQL client. (Replace the `<>` parts with values from the output):

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

> **Note:**
> You need to install the MySQL client before you connect to TiDB via MySQL.

5.5.5 Interact with the GKE cluster

You can interact with the GKE cluster by using `kubectl` and `helm` with the `credentials` ↪ `/kubeconfig_<gke_cluster_name>` kubeconfig file in the following two ways.

> **Note:**
>
> The default `gke_cluster_name` is `tidb-cluster`, which can be modified by changing `gke_name` in the `variables.tf` file.

- Specify the `--kubeconfig` option:

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

> **Note:**
>
> The `--kubeconfig` option used by the following command requires Helm 2.10.0 or later versions.

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

- Set the `KUBECONFIG` environment variable:

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

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

```
helm ls
```

5.5.6 Upgrade the TiDB cluster

To upgrade the TiDB cluster, perform the following steps:

1. Modify the `tidb_version` variable to a higher version in the `variables.tf` file.
2. Run `terraform apply`.

For example, to upgrade the cluster to the 3.0.0-rc.2 version, modify the `tidb_version` to `v3.0.0-rc.2`:

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

The upgrading does not finish immediately. You can run `kubectl --kubeconfig ↪ credentials/kubeconfig_<gke_cluster_name> get po -n tidb --watch` to verify that all pods are in `Running` state. Then you can access the database and use `tidb_version ↪ ()` to see whether the cluster has been upgraded successfully:

```sql
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 Manage multiple TiDB clusters

An instance of a `tidb-cluster` module corresponds to a TiDB cluster in the GKE cluster. To add a new TiDB cluster, perform the following steps:

1. Edit the `tidbclusters.tf` file and add a `tidb-cluster` module.

   For example:

   ```
   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
}
```

Note:

- `cluster_name` must be unique for each cluster.
- The total number of nodes actually created for each component is equal to the number of nodes in the configuration file multiplied by the number of Availability Zones in the region.

You can use `kubectl` to get the addresses for the TiDB cluster created and its monitoring service. If you want the Terraform script to print this information, add an `output` section in `outputs.tf` as follows:

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

This above configuration enables this script to print out the exact command used to connect to the TiDB cluster.

2. After you finish modification, execute `terraform init` and `terraform apply` to create the cluster.

5.5.8 Scale the TiDB cluster

To scale the TiDB cluster, perform the following steps:

1. Modify the `tikv_count` or `tidb_count` variable in the `variables.tf` file to your desired count.
2. Run `terraform apply`.

> **Warning:**
>
> Currently, scaling in is not supported because it cannot be determined which node will be removed. Scaling in by modifying `tikv_count` can lead to data loss.

Scaling out needs a few minutes to complete, you can watch the scaling-out process by running the following command:

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

For example, to scale out the cluster, you can modify the number of TiDB instances (`tidb_count`) from 1 to 2:

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

> **Note:**
>
> Incrementing the node count creates a node per GCP Availability Zone.

5.5.9 Customize

While you can change the default values in the `variables.tf` file, such as the cluster name or image version, it is recommended that you specify values in `terraform.tfvars` or another file of your choice.

5.5.9.1 Customize GCP resources

In GCP, you can attach a local SSD to any instance type that is `n1-standard-1` or greater, which provides good customizability.

5.5.9.2 Customize TiDB parameters

The Terraform scripts provide proper default settings for the TiDB cluster in GKE. You can also specify `override_values` or `override_values_file` variables in the `tidbclusters ↪ .tf` file for each TiDB cluster. If both variables are configured, then `override_values` is enabled and overrides the default settings. For example:

```
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"
```

By default, the cluster uses `values/default.yaml` in the `deploy/modules/gcp/tidb-` ↪ `cluster` module as the overriding values file.

In GKE, some configuration items are not customizable in `values.yaml`, such as the cluster version, the number of replicas, `NodeSelector`, and `Tolerations`. `NodeSelector` and `Tolerations` are controlled by Terraform to ensure consistency between the infrastructure and TiDB clusters.

To customize the cluster version and the number of replicas, directly modify arguments of the `tidb-cluster` module in the `clusters.tf` file.

> **Note:**
>
> It is not recommended to include the following configurations (default configurations of the `tidb-cluster` module) in the customized `values.yaml`.

```
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 Customize TiDB Operator

You can customize TiDB Operator by specifying overriding values through the `operator_helm_values` variable or specifying an overriding values file through the `operator_helm_values_file` variable. If both variables are configured, then `operator_helm_values` ↪ will be enabled and its value will be passed into the `tidb-cluster` module.

```
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 Customize logging

GKE uses Fluentd as its default log collector, which then forwards logs to Stackdriver. The Fluentd process can be quite resource hungry and consume a non-trivial share of CPU and RAM. Fluent Bit is a more performant and less resource intensive alternative. It is recommended to use Fluent Bit over Fluentd for a production set up. See this repository for an example of how to set up Fluent Bit on a GKE cluster.

5.5.9.5 Customize node pools

The cluster is created as a regional, as opposed to a zonal cluster. This means that GKE replicates node pools to each Availability Zone. This is desired to maintain high availability, however, for the monitoring services, like Grafana, this is potentially unnecessary. It is possible to manually remove nodes if desired via `gcloud`.

> **Note:**
>
> GKE node pools are managed instance groups, so a node deleted by `gcloud compute instances delete` will be automatically recreated and added back to the cluster.

Suppose that you need to delete a node from the monitor pool. You can perform the following steps:

1. Get the managed instance group and the Available Zone.

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

   The output is something like this:

   ```
   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
   ```

   The first column is the name of the managed instance group, and the second column is the Available Zone where it is created.

2. Get the name of the instance in that instance group.

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

   For example:

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

   The output is something like this:

   ```
   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. Delete the instance by specifying the name of the managed instance group and the name of the instance.

   For example,

```
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 Destroy a TiDB cluster

When you are done, the infrastructure can be torn down by running the following command:

```
terraform destroy
```

Note:

When `terraform destroy` is running, an error with the following message might occur: `Error reading Container Cluster "tidb": Cluster ↪ "tidb" has status "RECONCILING" with message""`. This happens when GCP is upgrading the Kubernetes master node, which it does automatically at times. While this is happening, it is not possible to delete the cluster. When it is done, run `terraform destroy` again.

5.5.10.1 Delete disks after use

If you no longer need the data and would like to delete the disks in use, you can choose one of the following two ways:

- Manual deletion: do this either in Google Cloud Console or using the `gcloud` command-line tool.
- Setting the Kubernetes persistent volume reclaiming policy to `Delete` prior to executing `terraform destroy`: Do this by running the following `kubectl` command before `terraform destroy`.

```
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"}}'
```

This command gets the persistent volume claims (PVCs) in the TiDB cluster namespace and sets the reclaiming policy of the persistent volumes to `Delete`. When the PVCs are deleted during `terraform destroy` execution, the disks are deleted as well.

The following `change-pv-reclaimpolicy.sh` script simplifies the above process in the `deploy/gcp` directory comparing to the root directory of the repository.

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

5.5.11 Manage multiple Kubernetes clusters

This section describes the best practices for managing multiple Kubernetes clusters, each with one or more TiDB clusters installed.

The Terraform module in TiDB typically combines the following sub-modules:

- `tidb-operator`: provisions the Kubernetes Control Plane and TiDB Operator for TiDB clusters
- `tidb-cluster`: creates the resource pool in the target Kubernetes cluster and deploys the TiDB cluster
- A `vpc` module, a `bastion` module, and a `project-credentials` module: dedicated to TiDB clusters on GKE

The best practices for managing multiple Kubernetes clusters are as follows:

- Creating a new directory for each of your Kubernetes clusters.
- Combining the above modules according to your needs via Terraform scripts.

If you use the best practices, the Terraform states among clusters do not interfere with each other, and it is convenient to manage multiple Kubernetes clusters. Here's an example (assume you are in the project root directory):

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

The content of `deploy/gcp-staging/main.tf` could be:

```
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: enforces Helm to depend on the GKE cluster
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
}
```

As shown in the code above, you can omit several parameters in each of the module calls because there are reasonable defaults, and it is easy to customize the configuration. For example, just delete the bastion module call if you do not need it.

To customize a field, use one of the following two methods:

- Modify the parameter configuration of `module` in the `*.tf` file directly.
- Refer to the `variables.tf` file of each module for all the modifiable parameters and set custom values in `terraform.tfvars`.

Note:

- When creating a new directory, pay attention to its relative path to Terraform modules, which affects the `source` parameter during module calls.
- If you want to use these modules outside the tidb-operator project, make sure you copy the whole `modules` directory and keep the relative path of each module inside the directory unchanged.
- Due to limitation hashicorp/terraform#2430 of Terraform, the `# HACK: ↪ enforces Helm to depend on the GKE cluster` section is added in the above example to deal with the Helm provider. If you write your own `tf` file, you need to include this section.

If you are unwilling to write Terraform code, you can also copy the `deploy/gcp` directory to create new Kubernetes clusters. But note that do not copy a directory that you have already run `terraform apply` against. In this case, it is recommended that you re-clone the tidb-operator repository before copying the directory.

5.6 Deploy TiDB on Alibaba Cloud Kubernetes

This document describes how to deploy a TiDB cluster on Alibaba Cloud Kubernetes with your laptop (Linux or macOS) for development or testing.

5.6.1 Prerequisites

- aliyun-cli >= 3.0.15 and configure aliyun-cli

> **Note:**
> The access key must be granted permissions to control the corresponding resources.

- kubectl >= 1.12
- helm >= 2.11.0 and < 2.16.4
- jq >= 1.6
- terraform 0.12.*

You can use Cloud Shell of Alibaba Cloud to perform operations. All the tools have been pre-installed and configured in the Cloud Shell of Alibaba Cloud.

5.6.1.1 Required privileges

To deploy a TiDB cluster, make sure you have the following privileges:

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

5.6.2 Overview of things to create

In the default configuration, you will create:

- A new VPC
- An ECS instance as the bastion machine
- A managed ACK (Alibaba Cloud Kubernetes) cluster with the following ECS instance worker nodes:
  - An auto-scaling group of 2 * instances (2 cores, 2 GB RAM) as ACK mandatory workers for the system service like CoreDNS
  - An auto-scaling group of 3 * `ecs.g5.large` instances for deploying the PD cluster
  - An auto-scaling group of 3 * `ecs.i2.2xlarge` instances for deploying the TiKV cluster
  - An auto-scaling group of 2 * `ecs.c5.4xlarge` instances for deploying the TiDB cluster
  - An auto-scaling group of 1 * `ecs.c5.xlarge` instance for deploying monitoring components
  - A 100 GB cloud disk used to store monitoring data

All the instances except ACK mandatory workers are deployed across availability zones (AZs) to provide cross-AZ high availability. The auto-scaling group ensures the desired number of healthy instances, so the cluster can auto-recover from node failure or even AZ failure.

5.6.3 Deploy

1. Configure the target region and Alibaba Cloud key (you can also set these variables in the `terraform` command prompt):

```
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>
```

   The `variables.tf` file contains default settings of variables used for deploying the cluster. You can change it or use the `-var` option to override a specific variable to fit your need.

2. Use Terraform to set up the cluster.

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

   Note that you must answer "yes" to `terraform apply` to continue:

```
terraform init
```

```
terraform apply
```

If you get an error while running `terraform apply`, fix the error (for example, lack of permission) according to the error description and run `terraform apply` again.

It takes 5 to 10 minutes to create the whole stack using `terraform apply`. Once installation is complete, the basic cluster information is printed:

```
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
```

Note:
You can use the `terraform output` command to get the output again.

3. You can then interact with the ACK cluster using `kubectl` or `helm`

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

```
kubectl version
```

```
helm ls
```

5.6.4 Access the database

You can connect the TiDB cluster via the bastion instance. All necessary information is in the output printed after installation is finished (replace the `<>` parts with values from the output):

```
ssh -i credentials/<cluster_name>-key.pem root@<bastion_ip>
```

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

5.6.5 Monitor

Visit `<monitor_endpoint>` to view the Grafana dashboards. You can find this information in the output of installation.

The initial login user account and password:

- User: admin
- Password: admin

> **Warning:**
>
> It is strongly recommended to set `deploy/modules/aliyun/tidb-cluster/values/default.yaml - monitor.grafana.service.annotations - service.beta.kubernetes.io/alicloud-loadbalancer-address-type` to `intranet` for security if you already have a VPN connecting to your VPC or plan to set up one.

5.6.6 Upgrade

To upgrade the TiDB cluster, set the `tidb_version` variable to a higher version in `variables.tf` and run `terraform apply`.

This may take a while to complete. You can watch the process using the following command:

```
kubectl get pods --namespace <tidb_cluster_name> -o wide --watch
```

5.6.7 Scale

To scale the TiDB cluster, modify `tikv_count` or `tidb_count` to your desired numbers, and then run `terraform apply`.

5.6.8 Configure

5.6.8.1 Configure TiDB Operator

You can adjust the `variables.tf` settings to configure TiDB Operator. Note that the `operator_helm_values` configuration item can provide a customized `values.yaml` configuration file for TiDB Operator. For example,

- Set `operator_helm_values` in `terraform.tfvars`:

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

- Set `operator_helm_values` in `main.tf`:

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

In the default configuration, the Terraform script creates a new VPC. To use the existing VPC, set `vpc_id` in `variable.tf`. In this case, Kubernetes nodes are not deployed in AZs with vswitch not configured.

5.6.8.2 Configure the TiDB cluster

`./my-cluster.yaml` is the `values.yaml` configuration file in the TiDB cluster. You can configure the TiDB cluster by modifying this file. For supported configuration items, see Configure the TiDB cluster in Kubernetes.

5.6.9 Manage multiple TiDB clusters

To manage multiple TiDB clusters in a single Kubernetes cluster, you need to edit `./` ↪ `main.tf` and add the `tidb-cluster` declaration based on your needs. For example:

```
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")
}
```

> **Note:**
>
> You need to set a unique `cluster_name` for each TiDB cluster.

All the configurable parameters in `tidb-cluster` are as follows:

Parameter	Description	Default value
`ack`	The structure that enwraps the target Kubernetes cluster information (required)	`nil`

Parameter	Description	Default value
`cluster_name`	The TiDB cluster name (required and unique)	`nil`
`tidb_version`	The TiDB cluster version	`v3.0.1`
`tidb_cluster_chart_version`	`tidb-cluster` helm chart version	`v1.0.1`
`pd_count`	The number of PD nodes	3
`pd_instance_type`	The PD instance type	`ecs.g5.large`
`tikv_count`	The number of TiKV nodes	3
`tikv_instance_type`	The TiKV instance type	`ecs.i2.2xlarge`

Parameter	Description	Default value
`tidb_count`	The number of TiDB nodes	2
`tidb_instance_type`	The TiDB instance type	`c5.4xlarge`
`monitor_instance_type`	The instance type of monitoring components	`c5.xlarge`

Parameter	Description	Default value
`override_values`	The `values.yaml` configuration file of the TiDB cluster. You can read it using the `file()` function	`nil`
`local_exec_interpreter`	The interpreter that executes the command line instruction	`["/bin/sh", "-c"]`

5.6.10 Manage multiple Kubernetes clusters

It is recommended to use a separate Terraform module to manage a specific Kubernetes cluster. (A Terraform module is a directory that contains the `.tf` script.)

`deploy/aliyun` combines multiple reusable Terraform scripts in `deploy/modules`. To manage multiple clusters, perform the following operations in the root directory of the `tidb` ↪ `-operator` project:

1. Create a directory for each cluster. For example:

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

2. Refer to `main.tf` in `deploy/aliyun` and write your own script. For example:

```
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
}
```

You can customize this script. For example, you can remove the `module "bastion"` declaration if you do not need the bastion machine.

> **Note:**
>
> You can copy the `deploy/aliyun` directory. But you cannot copy a directory on which the `terraform apply` operation is currently performed. In this case, it is recommended to clone the repository again and then copy it.

5.6.11 Destroy

It may take a long time to finish destroying the cluster.

```
terraform destroy
```

If you fail to create a Kubernetes cluster, an error is reported and you cannot clean the cluster normally when you try to destroy the cluster. In this case, you need to manually remove the Kubernetes resources from the local state and proceed to destroy the rest resources:

```
terraform state list
```

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

> **Note:**
>
> You have to manually delete the cloud disk used by monitoring node in the Alibaba Cloud console after destroying if you do not need it anymore.

5.6.12 Limitation

You cannot change `pod cidr`, `service cidr` and worker instance types once the cluster is created.

5.7 Access the TiDB Cluster in Kubernetes

This document describes how to access the TiDB cluster in Kubernetes.

- To access the TiDB cluster within a Kubernetes cluster, use the TiDB service domain name `<release-name>-tidb.<namespace>`.
- To access the TiDB cluster outside a Kubernetes cluster, expose the TiDB service port by editing the `tidb.service` field configuration in the `values.yaml` file of the `tidb-cluster` Helm chart.

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

5.7.1 NodePort

If there is no LoadBalancer, expose the TiDB service port in the following two modes of NodePort:

- `externalTrafficPolicy=Cluster`: All machines in the Kubernetes cluster assign a NodePort to TiDB Pod, which is the default mode.

  When using the `Cluster` mode, you can access the TiDB service by using the IP address of any machine plus a same port. If there is no TiDB Pod on the machine, the corresponding request is forwarded to the machine with a TiDB Pod.

> **Note:**
> In this mode, the request's source IP obtained by the TiDB server is the node IP, not the real client's source IP. Therefore, the access control based on the client's source IP is not available in this mode.

- `externalTrafficPolicy=Local`: Only those machines that runs TiDB assign NodePort to TiDB Pod so that you can access local TiDB instances.

When you use the Local mode, it is recommended to enable the StableScheduling feature of tidb-scheduler. tidb-scheduler tries to schedule the newly added TiDB instances to the existing machines during the upgrade process. With such scheduling, client outside the Kubernetes cluster does not need to upgrade configuration after TiDB is restarted.

5.7.1.1 View the IP/PORT exposed in NodePort mode

To view the Node Port assigned by Service, run the following commands to obtain the Service object of TiDB:

```
namespace=<your-tidb-namesapce>
```

```
release=<your-tidb-release-name>
```

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

To check you can access TiDB services by using the IP of what nodes, see the following two cases:

- When externalTrafficPolicy is configured as Cluster, you can use the IP of any node to access TiDB services.
- When externalTrafficPolicy is configured as Local, use the following commands to get the nodes where the TiDB instance of a specified cluster is located:

```
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

If Kubernetes is run in an environment with LoadBalancer, such as GCP/AWS platform, it is recommended to use the LoadBalancer feature of these cloud platforms by setting tidb ↪ .service.type=LoadBalancer.

See Kubernetes Service Documentation to know more about the features of Service and what LoadBalancer in the cloud platform supports.

5.8 Maintain TiDB Binlog

This document describes how to maintain TiDB Binlog of a TiDB cluster in Kubernetes.

5.8.1 Prerequisites

- Deploy TiDB Operator;
- Install Helm and configure it with the official PingCAP chart.

5.8.2 Enable TiDB Binlog of a TiDB cluster

TiDB Binlog is disabled in the TiDB cluster by default. To create a TiDB cluster with TiDB Binlog enabled, or enable TiDB Binlog in an existing TiDB cluster:

1. Modify the `values.yaml` file as described below:

   - Set `binlog.pump.create` to `true`.
   - Set `binlog.drainer.create` to `true`.
   - Set `binlog.pump.storageClassName` and `binlog.drainer.storageClassName` to an available `storageClass` in your Kubernetes cluster.
   - Set `binlog.drainer.destDBType` to your desired downstream storage as needed, which is explained in details below.

     TiDB Binlog supports three types of downstream storage:

     – PersistenceVolume: the default downstream storage. You can configure a large PV for `drainer` (by modifying `binlog.drainer.storage`) in this case.
     – MySQL compatible databases: enabled by setting `binlog.drainer.destDBType` to `mysql`. Meanwhile, you must configure the address and credential of the target database in `binlog.drainer.mysql`.
     – Apache Kafka: enabled by setting `binlog.drainer.destDBType` to `kafka`. Meanwhile, you must configure the zookeeper address and Kafka address of the target cluster in `binlog.drainer.kafka`.

2. Set affinity and anti-affinity for TiDB and the Pump component:

> **Note:**
> If you enable TiDB Binlog in the production environment, it is recommended to set affinity and anti-affinity for TiDB and the Pump component; if you enable TiDB Binlog in a test environment on the internal network, you can skip this step.

   By default, TiDB's affinity is set to `{}`. Currently, each TiDB instance does not have a corresponding Pump instance by default. When TiDB Binlog is enabled, if Pump and TiDB are separately deployed and network isolation occurs, and `ignore-error` is enabled, TiDB loses binlogs. In this situation, it is recommended to deploy a TiDB instance and a Pump instance on the same node using the affinity feature, and to split Pump instances on different nodes using the anti-affinity feature. For each node, only one Pump instance is required.

Note:

`<release-name>` needs to be replaced with the `Helm-release-name` of the target `tidb-cluster` chart.

- Configure `tidb.affinity` as follows:

```
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
```

- Configure `binlog.pump.affinity` as follows:

```
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. Create a new TiDB cluster or update an existing cluster:

   - Create a new TiDB cluster with TiDB Binlog enabled:

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

- Update an existing TiDB cluster to enable TiDB Binlog:

> Note:
> If you set the affinity for TiDB and its components, updating the existing TiDB cluster causes rolling updates of the TiDB components in the cluster.

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

5.8.3 Deploy multiple drainers

By default, only one downstream drainer is created. You can install the `tidb-drainer` Helm chart to deploy more drainers for a TiDB cluster, as described below:

1. Make sure that the PingCAP Helm repository is up to date:

```
helm repo update
```

```
helm search tidb-drainer -l
```

2. Get the default `values.yaml` file to facilitate customization:

```
helm inspect values pingcap/tidb-drainer --version=<chart-version> >
    ↪ values.yaml
```

3. Modify the `values.yaml` file to specify the source TiDB cluster and the downstream database of the drainer. Here is an example:

```
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
```

The `clusterName` and `clusterVersion` must match the desired source TiDB cluster.

For complete configuration details, refer to TiDB Binlog Drainer Configurations in Kubernetes.

4. Deploy the drainer:

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

> **Note:**
> This chart must be installed to the same namespace as the source TiDB cluster.

6 Configure

6.1 Initialize a TiDB Cluster in Kubernetes

This document describes how to initialize a TiDB cluster in Kubernetes (K8s), specifically, how to configure the initial account and password and how to initialize the database by executing SQL statements automatically in batch.

> **Note:**
> The following steps only apply when you create a cluster for the first time. Further configuration or modification after the initial cluster creation is not valid.

6.1.1 Set initial account and password

When a cluster is created, a default account `root` is created with no password. This might cause security issues. You can set a password for the `root` account in the following steps:

1. Create the `Namespace`.

   Before creating the cluster, create the Namespace:

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

2. Create a `secret` object.

   Before creating a cluster, create a `secret` to specify the password for `root`:

   ```
   kubectl create secret generic tidb-secret --from-literal=root=<root-
       ↪ password> --namespace=<namespace>
   ```

   If you also want to create users automatically, append the desired user name and the password, for example:

   ```
   kubectl create secret generic tidb-secret --from-literal=root=<root-
       ↪ password> --from-literal=developer=<developer-passowrd> --
       ↪ namespace=<namespace>
   ```

   This command creates users `root` and `developer` with their passwords, which are saved in the `tidb-secret` object. By default, the regular user `developer` is only granted with `USAGE` privilege; other privileges are set in the configuration item `tidb.initSql`.

3. Set a host that has access to TiDB.

   Before deploying the cluster, you can set a host that has access to TiDB by using the `tidb.permitHost` configuration item. If it is not set, all hosts have access to TiDB. For details, refer to Mysql GRANT host name.

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

4. Deploy the cluster.

   After creating the `secret`, deploy the cluster using the following command:

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

   After specifying `tidb.passwordSecretName`, the above command sets up a cluster with an initialization job created automatically. Using the available `secret`, this job creates the password for the `root` account, and creates other user accounts and passwords if specified. The password specified here is required when you login to the MySQL client.

Note:
When the initialization job is created, the Pod for the TiDB cluster has not been created fully. There might be a few errors before initialization completes and Pod state becomes Completed.

6.1.2 Initialize SQL statements in batch

You can also execute the SQL statements in batch in `tidb.initSql` for initialization. This function by default creates some databases or tables for the cluster and performs user privilege management operations. For example, the following configuration automatically creates a database named `app` after the cluster creation, and grants the `developer` account full management privileges on `app`.

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

Save the above configuration to the `values.yaml` file and run the following command to deploy the cluster:

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

> **Note:**
>
> Currently no verification has been implemented for `initSql`. You can create accounts and set passwords in `initSql`, but it is not recommended because passwords created this way are saved as plaintext in the initializer job object.

6.2 TiDB Cluster Configurations in Kubernetes

This document introduces the following items of a TiDB cluster in Kubernetes:

- The configuration parameters
- The configuration of resources
- The configuration of disaster recovery

6.2.1 Configuration parameters

TiDB Operator uses `Helm` to deploy and manage TiDB clusters. The configuration file obtained through Helm provides the basic configuration by default with which you could quickly start a TiDB cluster. However, if you want special configurations or are deploying in a production environment, you need to manually configure the corresponding parameters according to the table below.

Note:

In the following table, `values.yaml` refers to the TiDB cluster's configuration file to be modified.

Parameter	Description	Default Value
`rbac.create`	Whether to enable the RBAC mode of Kubernetes	`true`

Parameter	Description	Default Value
`clusterName`	The TiDB cluster name. This variable is unset by default. In this case, `tidb-cluster` directly replaces it with `ReleaseName` when the cluster is being installed.	`nil`
`extraLabels`	Adds extra labels to the `TidbCluster` object (CRD). See labels	`{}`

Parameter	Description	Default Value
`schedulerName`	The scheduler used by the TiDB cluster	`tidb-scheduler`
`timezone`	The default timezone used by the TiDB cluster	`UTC`
`pvReclaimPolicy`	The reclaim policy for PV (Persistent Volume) used by the TiDB cluster	`Retain`

Parameter	Description	Default Value
`services[0].name`	The name of the service that the TiDB cluster exposes	`nil`
`services[0].type`	The type of the service that the TiDB cluster exposes (selected from `ClusterIP`, `NodePort` and `LoadBalancer`)	`nil`

Parameter	Description	Default Value
`discovery.image`	The image of PD's service discovery component in the TiDB cluster. This component is used to provide service discovery for each PD instance to coordinate the starting sequence when the PD cluster is started f	`pingcap/tidb-operator:v1.0.0`

Parameter	Description	Default Value
`discovery.imagePullPolicy`	The pulling policy for the image of PD's service discovery component	`IfNotPresent`
`discovery.resources.limits.cpu`	The CPU resource limit of PD's service discovery component	`250m`

Parameter	Description	Default Value
`discovery.resources.limits.memory`	The memory resource limit of PD's service discovery component	`150Mi`
`discovery.resources.requests.cpu`	The CPU resource request of PD's service discovery component	`80m`

Parameter	Description	Default Value
`discovery.resources.requests.memory`	The memory resource request of PD's service discovery component	`50Mi`

Parameter	Description	Default Value
`enableConfigMapRollout`	Whether to enable the automatic rolling update of the TiDB cluster. If enabled, the TiDB cluster automatically updates the corresponding components when the `ConfigMap` of this cluster changes. This configuration is only supported in	`false`

Parameter	Description	Default Value
`pd.config`	The configuration of PD. Check [the `config.toml` file](#) for the default PD configuration file (by choosing the tag of the corresponding PD version). You can see [PD Configuration Flags](#) for the detailed descrip-	If the version of TiDB Operator is v1.0.0 or earlier, the default value is `nil`If the version of TiDB Operator is later than v1.0.0, the default value is `[log] level = "info" [replication] location-labels = ["region", "zone", "rack", "`

Parameter	Description	Default Value
`pd.replicas`	The number of Pods in PD	`3`
`pd.image`	The PD image	`pingcap/pd:v3.0.0-rc.1`
`pd.imagePullPolicy`	The pulling policy for the PD image	`IfNotPresent`

Parameter	Description	Default Value
`pd.logLevel`	The log level of PDIf the version of TiDB Operator is later than v1.0.0, configure the parameter via `pd.config`: `[log] level = "info"`	`info`

Parameter	Description	Default Value
`pd.storageClassName`	The `storageClassName` used by PD. `storageClassName` refers to a type of storage provided by the Kubernetes cluster, which might map to a level of service quality, a backup policy, or to any policy determined by the cluster administrator. Detailed	`local-storage`

Parameter	Description	Default Value
`pd.maxStoreDownTime`	This parameter indicates how soon a store node is marked as `down` after it is disconnected. When the state changes to `down`, the store node starts migrating data to other store nodes.If the version of TiDB Operator is later than v1.0.0,	`30m`

Parameter	Description	Default Value
`pd.maxReplicas`	The number of data replicas in the TiDB clusterIf the version of TiDB Operator is later than v1.0.0, configure this parameter via `pd.config: [replication] max-replicas = 3`	3

Parameter	Description	Default Value
`pd.resources.limits.cpu`	The CPU resource limit per PD Pod	`nil`
`pd.resources.limits.memory`	The memory resource limit per PD Pod	`nil`
`pd.resources.limits.storage`	The storage limit per PD Pod	`nil`
`pd.resources.requests.cpu`	The CPU resource requests of each PD Pod	`nil`

Parameter	Description	Default Value
`pd.resources.requests.memory`	The memory resource requests of each PD Pod	`nil`
`pd.resources.requests.storage`	The storage requests of each PD Pod	`1Gi`
`pd.affinity`	Defines PD's scheduling rules and preferences. Detailed reference: affinity-and-anti-affinity	`{}`

Parameter	Description	Default Value
`pd.nodeSelector`	Ensures that PD Pods are only scheduled to the node with the specific key-value pair as the label. Detailed reference: nodeSelector	`{}`

Parameter	Description	Default Value
`pd.tolerations`	Applies to PD Pods, allowing the Pods to be scheduled to the nodes with specified taints. Detailed reference: taint-and-toleration	{}
`pd.annotations`	Adds specific `annotations` for PD Pods.	{}

Parameter	Description	Default Value
`tikv.config`	The configuration of TiKV. Check [the `config-template.toml` file](#) for the default TiKV configuration file (by choosing the tag of the corresponding TiKV version). You can see [TiKV Configuration Flags](#) for the detailed [illegible]	If the version of TiDB Operator is v1.0.0-beta.3 or earlier, the default value is `nil` If the version of TiDB Operator is later than v1.0.0-beta.3, the default value is `log-level = "info"` Sample configuration: `config: | log-level = "info"`

Parameter	Description	Default Value
`tikv.replicas`	The number of Pods in TiKV	`3`
`tikv.image`	The TiKV image	`pingcap/tikv:v3.0.0-rc.1`
`tikv.imagePullPolicy`	The pulling policy for the TiKV image	`IfNotPresent`

Parameter	Description	Default Value
`tikv.logLevel`	The level of TiKV logsIf the version of TiDB Operator is later than v1.0.0, configure this parameter via `tikv.config:log-level = "info"`	`info`

Parameter	Description	Default Value
`tikv.storageClassName`	The `storageClassName` used by TiKV. `storageClassName` refers to a type of storage provided by the Kubernetes cluster, which might map to a level of service quality, a backup policy, or to any policy determined by the cluster administrator. Detailed	`local-storage`

Parameter	Description	Default Value
`tikv.syncLog`	`SyncLog` means whether to enable the raft log replication. Enabling this feature ensures that data will not be lost when power is off.If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tikv.config`	`true`

Parameter	Description	Default Value
`tikv.grpcConcurrency`	Configures the thread pool size of the gRPC server.If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tikv.config:[server]grpc-concurrency = 4`	4

Parameter	Description	Default Value
`tikv.resources.limits.cpu`	The CPU resource limit per TiKV Pod	`nil`
`tikv.resources.limits.memory`	The memory resource limit per TiKV Pod	`nil`
`tikv.resources.limits.storage`	The storage limit per TiKV Pod	`nil`
`tikv.resources.requests.cpu`	The CPU resource requests of each TiKV Pod	`nil`

Parameter	Description	Default Value
`tikv.resources.requests.memory`	The memory resource requests of each TiKV Pod	`nil`
`tikv.resources.requests.storage`	The storage requests of each TiKV Pod	`10Gi`
`tikv.affinity`	Defines TiKV's scheduling rules and preferences. Detailed reference:affinity-and-anti-affinity	`{}`

Parameter	Description	Default Value
`tikv.nodeSelector`	Ensures that TiKV Pods are only scheduled to the node with the specific key-value pair as the label. Detailed reference: nodeSelector	`{}`

Parameter	Description	Default Value
`tikv.tolerations`	Applies tolerations to TiKV Pods, allowing TiKV Pods to be scheduled to the nodes with specified taints. Detailed reference: taint-and-toleration	{}
`tikv.annotations`	Adds specific `annotations` for TiKV Pods.	{}

Parameter	Description	Default Value
`tikv.defaultcfBlockCacheSize`	Specifies the size of block cache which is used to cache uncompressed blocks. Larger block cache settings speed up reads. It is recommended to set the parameter to 30%-50% of the value of `tikv.resources.limits.memory`. If the version of TiDB O	`1GB`

Parameter	Description	Default Value
`tikv.writecfBlockCacheSize`	Specifies the size of writecf block cache. It is recommended to set the parameter to 10%-30% of the value of `tikv.resources.limits.memory`. If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tikv.config:[rocksdb`	`256MB`

Parameter	Description	Default Value
`tikv.readpoolStorageConcurrency`	The size of thread pool for high priority, normal priority or low priority operations in the TiKV storageIf the version of TiDB Operator is later than v1.0.0, configure this parameter via `tikv.config:[readpool.storage]high-`	4

Parameter	Description	Default Value
`tikv.readpoolCoprocessorConcurrency`	If `tikv.resources.limits.cpu` is greater than 8, set the value of `tikv.readpoolCoprocessorConcurrency` to `tikv.resources.limits.cpu` * 0.8If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tikv.config:[readpool.`	8

Parameter	Description	Default Value
`tikv.storageSchedulerWorkerPoolSize`	The worker pool size of the TiKV scheduler. This size must be increased in the case of rewriting but be smaller than the total CPU cores.If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tikv.config:[storage]`	4

Parameter	Description	Default Value
`tidb.config`	The configuration of TiDB. Check [the `config.toml.example` file](#) for the default TiDB configuration file (by choosing the tag of the corresponding TiDB version). You can see [TiDB Configuration File Description](#) for th	If the version of TiDB Operator is v1.0.0-beta.3 or earlier, the default value is `nil` If the version of TiDB Operator is later than v1.0.0-beta.3, the default value is `[log] level = "info"` Sample configuration: `config: | [log] level = "info"`

Parameter	Description	Default Value
`tidb.replicas`	The number of Pods in TiDB	2
`tidb.image`	The TiDB image	`pingcap/tidb:v3.0.0-rc.1`
`tidb.imagePullPolicy`	The pulling policy for the TiDB image	`IfNotPresent`

Parameter	Description	Default Value
`tidb.logLevel`	The level of TiDB logsIf the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config`: `[log] level = "info"`	`info`
`tidb.resources.limits.cpu`	The CPU resource limit per TiDB Pod	`nil`

Parameter	Description	Default Value
`tidb.resources.limits.memory`	The memory resource limit per TiDB Pod	`nil`
`tidb.resources.requests.cpu`	The CPU resource requests of each TiDB Pod	`nil`
`tidb.resources.requests.memory`	The memory resource requests of each TiDB Pod	`nil`

Parameter	Description	Default Value
tidb.passwordSecretName	The name of the Secret that stores the TiDB username and password. The Secret can create a secret with this command: kubectl create secret generic tidb secret --from literal=root=<root password>--	nil

Parameter	Description	Default Value
`tidb.initSql`	The initialization script that will be executed after a TiDB cluster is successfully started.	`nil`
`tidb.affinity`	Defines TiDB's scheduling rules and preferences. Detailed reference: affinity-and-anti-affinity	`{}`

Parameter	Description	Default Value
`tidb.nodeSelector`	Ensures that TiDB Pods are only scheduled to the node with the specific key-value pair as the label. Detailed reference: nodeSelector	`{}`

Parameter	Description	Default Value
`tidb.tolerations`	Applies to TiDB Pods, allowing TiDB Pods to be scheduled to nodes with specified taints. Detailed reference: taint-and-toleration	`{}`
`tidb.annotations`	Adds specific `annotations` for TiDB Pods.	`{}`

Parameter	Description	Default Value
`tidb.maxFailoverCount`	The maximum number of failovers for TiDB. Assuming the number is `3`, that is, up to 3 failovers TiDB instances are supported at the same time.	`3`
`tidb.service.type`	The type of service that the TiDB cluster exposes	`Nodeport`

Parameter	Description	Default Value
`tidb.service.externalTrafficPolicy`	Whether this Service routes external traffic to a node-local or cluster-wide endpoint. There are two options available: `Cluster` (by default) and `Local`. `Cluster` obscures the client source IP and some traffic needs to hop twice among nodes for	`nil`

Parameter	Description	Default Value
`tidb.service.loadBalancerIP`	Specifies the IP of LoadBalancer. Some cloud providers allow you to specify `loadBalancerIP`. In these cases, the LoadBalancer will be created using the user-specified `loadBalancerIP`. If the `loadBalancerIP` field is not specified, the LoadBalancer will be set using the tem-	`nil`

Parameter	Description	Default Value
`tidb.service.mysqlNodePort`	The MySQL `NodePort` that TiDB Service exposes	
`tidb.service.exposeStatus`	The port that indicates the expose status of TiDB Service	`true`
`tidb.service.statusNodePort`	The `NodePort` exposed through specifying the status of TiDB Service	

Parameter	Description	Default Value
`tidb.separateSlowLog`	Whether to run in the sidecar mode the `SlowLog` of TiDB exported by the independent container	If the version of TiDB Operator is v1.0.0 or earlier, the default value is `false`. If the version of TiDB Operator is later than v1.0.0, the default value is `true`.

Parameter	Description	Default Value
`tidb.slowLogTailer.image`	The image of TiDB's `slowLogTailer`. `slowLogTailer` is a container of the sidecar type, used to export the `SlowLog` of TiDB. This configuration only takes effect when `tidb.separateSlowLog=true`.	`busybox:1.26.2`

Parameter	Description	Default Value
`tidb.slowLogTailer.resources.limits.cpu`	The CPU resource limit per TiDB Pod's `slowLogTailer`	`100m`
`tidb.slowLogTailer.resources.limits.memory`	The memory resource limit per TiDB Pod's `slowLogTailer`	`50Mi`
`tidb.slowLogTailer.resources.requests.cpu`	The requests of each TiDB Pod's `slowLogTailer` for CPU resources	`20m`

Parameter	Description	Default Value
`tidb.slowLogTailer.resources.requests.memory`	The requests of each TiDB Pod's `slowLogTailer` for memory resources	`5Mi`
`tidb.plugin.enable`	Whether to enable the TiDB plugin	`false`
`tidb.plugin.directory`	Specifies the directory where the TiDB plugin is located.	`/plugins`

Parameter	Description	Default Value
`tidb.plugin.list`	Specifies a list of plugins loaded on TiDB. The naming rules of Plugin ID: `plugin name-version`. For example: `'conn_limit-1'`.	[]

Parameter	Description	Default Value
`tidb.preparedPlanCacheEnabled`	Whether to enable TiDB's prepared plan cacheIf the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config`: `[prepared-plan-cache] enabled = false`	`false`

Parameter	Description	Default Value
`tidb.preparedPlanCacheCapacity`	The cache capacity of TiDB's prepared planIf the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config:[prepared-plan-cache] capacity = 100`	100

Parameter	Description	Default Value
`tidb.txnLocalLatchesEnabled`	Whether to enable the memory lock for transactions. It is recommended to enable the lock when there are many local transaction conflicts.If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config`:[	`false`

Parameter	Description	Default Value
tidb.txnLocalLatchesCapacity	The capacity of the transaction memory lock. The number of slots corresponding to Hash is automatically adjusted upward to an exponential multiple of 2. Each slot occupies 32 Bytes of memory. When the range of writing data i	10240000

Parameter	Description	Default Value
`tidb.tokenLimit`	The restrictions on TiDB to execute concurrent sessionsIf the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config:token-limit = 1000`	`1000`

Parameter	Description	Default Value
`tidb.memQuotaQuery`	The memory quota for TiDB queries, which is 32GB by default.If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config:mem-quota-query = 34359738368`	`34359738368`

Parameter	Description	Default Value
`tidb.checkMb4ValueInUtf8`	Controls whether to check the `mb4` characters when the character set is `utf8`. If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config`: `check-mb4-value-in-utf8 = true`	`true`

Parameter	Description	Default Value
`tidb.treatOldVersionUtf8AsUtf8mb4`	This parameter is used for upgrading compatibility. When it is set to `true`, `utf8` character set in the old table/column is treated as `utf8mb4`. If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.`	`true`

Parameter	Description	Default Value
`tidb.lease`	The lease time of TiDB Schema lease. It is highly risky to change this parameter. Therefore, it is not recommended to do so unless you know exactly what might be happening.If the version of TiDB Operator is later than v1.0.0, configure thi	`45s`

Parameter	Description	Default Value
`tidb.maxProcs`	The maximum available CPU cores. `0` represents the number of CPU on the machine or Pod.If the version of TiDB Operator is later than v1.0.0, configure this parameter via `tidb.config:[performance] max-procs = 0`	`0`

Parameter	Description	Default Value

6.2.2 Resource configuration

Before deploying a TiDB cluster, it is necessary to configure the resources for each component of the cluster depending on your needs. PD, TiKV and TiDB are the core service components of a TiDB cluster. In a production environment, their resource configurations must be specified according to component needs. Detailed reference: Hardware Recommendations.

To ensure the proper scheduling and stable operation of the components of the TiDB cluster in Kubernetes, it is recommended to set Guaranteed-level QoS by letting `limits` equal to `requests` when configuring resources. Detailed reference: Configure Quality of Service for Pods.

If you are using a NUMA-based CPU, you need to enable `Static`'s CPU management policy on the node for better performance. In order to allow the TiDB cluster component to monopolize the corresponding CPU resources, the CPU quota must be an integer greater than or equal to `1` besides setting Guaranteed-level QoS as mentioned above. Detailed reference: Control CPU Management Policies on the Node.

6.2.3 Disaster recovery configuration

TiDB is a distributed database and its disaster recovery must ensure that when any physical topology node fails, not only the service is unaffected, but also the data is complete and available. The two configurations of disaster recovery are described separately as follows.

6.2.3.1 Disaster recovery of TiDB service

The disaster recovery of TiDB service is essentially based on Kubernetes' scheduling capabilities. To optimize scheduling, TiDB Operator provides a custom scheduler that guarantees the disaster recovery of TiDB service at the host level through the specified scheduling algorithm. Currently, the TiDB cluster uses this scheduler as the default scheduler, which is configured through the item `schedulerName` in the above table.

Disaster recovery at other levels (such as rack, zone, region) are guaranteed by Affinity's `PodAntiAffinity`. `PodAntiAffinity` can avoid the situation where different instances of the same component are deployed on the same physical topology node. In this way, disaster recovery is achieved. Detailed user guide for Affinity: Affinity & AntiAffinity.

The following is an example of a typical disaster recovery setup:

```
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 Disaster recovery of data

Before configuring the data disaster recovery, read Information Configuration of the Cluster Typology which describes how the disaster recovery of the TiDB cluster is implemented.

To add the data disaster recovery feature in Kubernetes:

1. Set the label collection of topological location for PD

   Configure `location-labels` in the `pd.config` file using the labels that describe the topology on the nodes in the Kubernetes cluster.

> **Note:**
>
> - For PD versions $<$ v3.0.9, the `/` in the label name is not supported.
> - If you configure `hostname` in the `location-labels`, TiDB Operator will get the value from the `kubernetes.io/hostname` label on the node.

2. Set the topological information of the Node where the TiKV node is located.

   TiDB Operator automatically obtains the topological information of the Node for TiKV and calls the PD interface to set this information as the information of TiKV's store labels. Based on this topological information, the TiDB cluster schedules the replicas of the data.

   If the Node of the current Kubernetes cluster does not have a label indicating the topological location, or if the existing label name of topology contains `/`, you can manually add a label to the Node by running the following command:

```
kubectl label node <nodeName> region=<regionName> zone=<zoneName> rack
    ↪ =<rackName> kubernetes.io/hostname=<hostName>
```

   In the command above, `region`, `zone`, `rack`, and `kubernetes.io/hostname` are just examples. The name and number of the label to be added can be arbitrarily defined, as long as it conforms to the specification and is consistent with the labels set by `location-labels` in `pd.config`.

6.3 The Backup Configuration of TiDB in Kubernetetes

`tidb-backup` is a helm chart used for backing up and restoring TiDB clusters in Kubernetes. This document describes the configuration of `tidb-backup`.

6.3.1 Configuration

6.3.1.1 `mode`

- The operating mode
- Default: "backup"
- Options: `backup` (for backing up the data of a cluster) or `restore` (for restoring the data of a cluster)

6.3.1.2 `clusterName`

- The name of the target cluster
- Default: "demo"
- The name of the TiDB cluster from which data is backed up or to which data is restored

6.3.1.3 `name`

- The name of the backup
- Default: "fullbackup-{{ date"200601021504" .Release.Time }}". `date` is the starting time of the backup, which is accurate to minute.

6.3.1.4 `secretName`

- The name of the `Secret` which stores the credential of the target cluster. See Kubernetes Secret for reference.

- Default: "backup-secret"

- You can create the `Secret` by running the following command:

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

6.3.1.5 `storage.className`

- The StorageClass is used to store backup data.
- Default: "local-storage"
- The backup job needs a Persistent Volume (PV) to store the backup data. You must ensure that `StorageClass` is available in your Kubernetes cluster.

6.3.1.6 `storage.size`

- The storage size of the Persistence Volume
- Default: "100Gi"

6.3.1.7 `backupOptions`

- The optional parameter specified to `mydumper` used when backing up data
- Default: "–chunk-filesize=100"

6.3.1.8 `restoreOptions`

- The optional parameter specified to `loader` used when backing up data
- Default: "-t 16"

6.3.1.9 `gcp.bucket`

- The name of the GCP bucket used to store backup data
- Default: ""

> **Note:**
>
> Once you set any variable under the `gcp` section, the backup data will be uploaded to Google Cloud Storage, which means that you must keep the configuration intact.

6.3.1.10 `gcp.secretName`

- The name of the `Secret` that stores the credential of Google Cloud Storage

- Default: ""

- See Google Cloud Documentation to download the credential file and create the `Secret` by the running following command:

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

6.3.1.11 `ceph.endpoint`

- The endpoint of the `ceph` object storage
- Default: ""

> **Note:**
>
> Once you set any variable under the `ceph` section, the backup data will be uploaded to the `ceph` object storage, which means that you must keep the configuration intact.

6.3.1.12 `ceph.bucket`

- The bucket name of the `ceph` object storage
- Default: ""

6.3.1.13 `ceph.secretName`

- The name of the `Secret` that stores the credential of the `ceph` object store

- Default: ""

- You can create the `Secret` by running the following command:

```
kubectl create secret generic ceph-backup-secret -n <namespace> --from-
    ↪ literal=access_key=<access-key> --from-literal=secret_key=<secret
    ↪ -key>
```

6.4 Persistent Storage Class Configuration in Kubernetes

TiDB cluster components such as PD, TiKV, TiDB monitoring, TiDB Binlog and `tidb` ↪ `-backup` require the persistent storage of data. To persist the data in Kubernetes, you need to use PersistentVolume (PV). Kubernetes supports several types of storage classes, which are mainly divided into two parts:

- Network storage

  The network storage medium is not on the current node, but is mounted to the node through the network. Generally, there are redundant replicas to guarantee high availability. When the node fails, the corresponding network storage can be re-mounted to another node for further use.

- Local storage

  The local storage medium is on the current node, and typically can provide lower latency than the network storage. Because there are no redundant replicas, once the node fails, data might be lost. If it is an IDC server, data can be restored to a certain extent. If it is a virtual machine using the local disk on the public cloud, data **cannot** be retrieved after the node fails.

PVs are created automatically by the system administrator or volume provisioner. PVs and Pods are bound by PersistentVolumeClaim (PVC). Users request for using a PV through a PVC instead of creating a PV directly. The corresponding volume provisioner creates a PV that meets the requirements of PVC and then binds the PV to the PVC.

> **Warning:**
>
> For data safety, do not delete a PV in any case unless you are familiar with the underlying volume provisioner.

6.4.1 Recommended storage classes for TiDB clusters

TiKV uses the Raft protocol to replicate data. When a node fails, PD automatically schedules data to fill the missing data replicas; TiKV requires low read and write latency, so local SSD storage is strongly recommended in the production environment.

PD also uses Raft to replicate data. PD is not an I/O-intensive application, but a database for storing cluster meta information, so a local SAS disk or network SSD storage such as EBS General Purpose SSD (gp2) volumes on AWS or SSD persistent disks on GCP can meet the requirements.

To ensure availability, it is recommended to use network storage for components such as TiDB monitoring, TiDB Binlog and `tidb-backup` because they do not have redundant replicas. TiDB Binlog's Pump and Drainer components are I/O-intensive applications that require low read and write latency, so it is recommended to use high-performance network storage such as EBS Provisioned IOPS SSD (io1) volumes on AWS or SSD persistent disks on GCP.

When deploying TiDB clusters or `tidb-backup` with TiDB Operator, you can configure `StorageClass` for the components that require persistent storage via the corresponding `storageClassName` field in the `values.yaml` configuration file. The `StorageClassName` is set to `local-storage` by default.

6.4.2 Network PV configuration

Kubernetes 1.11 and later versions support volume expansion of network PV, but you need to run the following command to enable volume expansion for the corresponding `StorageClass`:

```
kubectl patch storageclass <storage-class-name> -p '{"allowVolumeExpansion":
    ↪  true}'
```

After volume expansion is enabled, expand the PV using the following method:

1. Edit the PersistentVolumeClaim (PVC) object:

   Suppose the PVC is 10 Gi and now we need to expand it to 100 Gi.

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

2. View the size of the PV:

   After the expansion, the size displayed by running `kubectl get pvc -n <namespace ↪ > <pvc-name>` is still the original one. But if you run the following command to view the size of the PV, it shows that the size has been expanded to the expected one.

```
kubectl get pv | grep <pvc-name>
```

6.4.3 Local PV configuration

Kubernetes currently supports statically allocated local storage. To create a local storage object, use `local-volume-provisioner` in the local-static-provisioner repository. The procedure is as follows:

1. Pre-allocate local storage in cluster nodes. See the operation guide provided by Kubernetes.

2. Deploy `local-volume-provisioner`.

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

   Check the Pod and PV status with the following commands:

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

   `local-volume-provisioner` creates a PV for each mounting point under discovery directory. Note that on GKE, `local-volume-provisioner` creates a local volume of only 375 GiB in size by default.

For more information, refer to Kubernetes local storage and local-static-provisioner document.

6.4.3.1 Best practices

- A local PV's path is its unique identifier. To avoid conflicts, it is recommended to use the UUID of the device to generate a unique path.
- For I/O isolation, a dedicated physical disk per PV is recommended to ensure hardware-based isolation.
- For capacity isolation, a partition per PV or a physical disk per PV is recommended.

Refer to Best Practices for more information on local PV in Kubernetes.

6.4.4 Disk mount examples

If the components such as monitoring, TiDB Binlog, and `tidb-backup` use local disks to store data, you can mount SAS disks and create separate `StorageClass` for them to use. Procedures are as follows:

- For a disk storing monitoring data, follow the steps to mount the disk. First, create multiple directories in disk, and bind mount them into `/mnt/disks` directory. Then, create `local-storage StorageClass` for them to use.

> **Note:**
> The number of directories you create depends on the planned number of TiDB clusters. For each directory, a corresponding PV will be created. The monitoring data in each TiDB cluster uses one PV.

- For a disk storing TiDB Binlog and backup data, follow the steps to mount the disk. First, create multiple directories in disk, and bind mount them into `/mnt/backup` directory. Then, create `backup-storage StorageClass` for them to use.

> **Note:**
> The number of directories you create depends on the planned number of TiDB clusters, the number of Pumps in each cluster, and your backup method. For each directory, a corresponding PV will be created. Each Pump uses one PV and each Drainer uses one PV. Each Ad-hoc full backup task uses one PV, and all scheduled full backup tasks share one PV.

- For a disk storing data in PD, follow the steps to mount the disk. First, create multiple directories in disk, and bind mount them into `/mnt/sharedssd` directory. Then, create `shared-ssd-storage StorageClass` for them to use.

> **Note:**
> The number of directories you create depends on the planned number of TiDB clusters, and the number of PD servers in each cluster. For each directory, a corresponding PV will be created. Each PD server uses one PV.

- For a disk storing data in TiKV, you can mount it into `/mnt/ssd` directory, and create `ssd-storage StorageClass` for it to use.

Based on the disk mounts above, you need to modify the `local-volume-provisioner` YAML file accordingly, configure discovery directory and create the necessary `StorageClass`. Here is an example of a modified YAML file:

```
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
......
```

Finally, execute the `kubectl apply` command to deploy `local-volume-provisioner`.

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

When you later deploy tidb clusters, deploy TiDB Binlog for incremental backups, or do full backups, configure the corresponding `StorageClass` for use.

6.4.5 Data safety

In general, after a PVC is no longer used and deleted, the PV bound to it is reclaimed and placed in the resource pool for scheduling by the provisioner. To avoid accidental data loss, you can globally configure the reclaim policy of the `StorageClass` to `Retain` or only change the reclaim policy of a single PV to `Retain`. With the `Retain` policy, a PV is not automatically reclaimed.

- Configure globally:

  The reclaim policy of a `StorageClass` is set at creation time and it cannot be updated once it is created. If it is not set when created, you can create another `StorageClass` of the same provisioner. For example, the default reclaim policy of the `StorageClass` for persistent disks on Google Kubernetes Engine (GKE) is `Delete`. You can create another `StorageClass` named `pd-standard` with its reclaim policy as `Retain`, and change the `storageClassName` of the corresponding component to `pd-standard` when creating a TiDB cluster.

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

- Configure a single PV:

  ```
  kubectl patch pv <pv-name> -p '{"spec":{"persistentVolumeReclaimPolicy
      ↪ ":"Retain"}}'
  ```

> **Note:**
>
> By default, to ensure data safety, TiDB Operator automatically changes the reclaim policy of the PVs of PD and TiKV to `Retain`.

When the reclaim policy of PVs is set to `Retain`, if the data of a PV can be deleted, delete this PV and the corresponding data according to the following steps:

1. Delete the PVC object corresponding to the PV:

```
kubectl delete pvc <pvc-name> --namespace=<namespace>
```

2. Set the reclaim policy of the PV to `Delete`. Then the PV is automatically deleted and reclaimed.

```
kubectl patch pv <pv-name> -p '{"spec":{"persistentVolumeReclaimPolicy
    ↪ ":"Delete"}}'
```

For more details, refer to Change the Reclaim Policy of a PersistentVolume.

6.5 TiDB Binlog Drainer Configurations in Kubernetes

This document introduces the configuration parameters for a TiDB Binlog drainer in Kubernetes.

6.5.1 Configuration parameters

The following table contains all configuration parameters available for the `tidb-drainer` chart. Refer to Use Helm to learn how to configure these parameters.

Parameter	Description	Default Value
`clusterName` ↪	The name of the source TiDB cluster	`demo`

Parameter	Description	Default Value
`clusterVersion`	The version of the source TiDB cluster	`v3.0.1`
`baseImage`	The base image of TiDB Binlog	`pingcap/tidb-binlog`
`imagePullPolicy`	The pulling policy of the image	`IfNotPresent`
`logLevel`	The log level of the drainer process	`info`

Parameter	Description	Default Value
`storageClassName`	`storageClassName` used by the drainer. `storageClassName` refers to a type of storage provided by the Kubernetes cluster, which might map to a level of service quality, a backup policy, or to any policy determined by the cluster administrator. Detailed refer-	`local-storage`

Parameter	Description	Default Value
`storage` ↪	The storage limit of the drainer Pod. Note that you should set a larger size if `db-` ↪ `type` ↪ is set to `pb`	`10Gi`
`disableDetect` ↪	Determines whether to disable casualty detection	`false`
`initialCommitTs` ↪	Used to initialize a checkpoint if the drainer does not have one	`0`

Parameter	Description	Default Value
`config`	The configuration file passed to the drainer. Detailed reference: drainer.toml	(see below)
`resources` ↪	The resource limits and requests of the drainer Pod	`{}`

Parameter	Description	Default Value
`nodeSelector`	Ensures that the drainer Pod is only scheduled to the node with the specific key-value pair as the label. Detailed reference: nodeselector	`{}`

Parameter	Description	Default Value
`tolerations`	Applies to drainer Pods, allowing the Pods to be scheduled to the nodes with specified taints. Detailed reference: taint-and-toleration	`{}`

Parameter	Description	Default Value
`affinity` ↪	Defines scheduling policies and preferences of the drainer Pod. Detailed reference: affinity-and-anti-affinity	{}

The default value of `config` is:

```
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 Monitor a TiDB Cluster in Kubernetes

Monitoring a TiDB cluster deployed in Kubernetes can be roughly divided into two parts:

- Monitoring the TiDB cluster itself
- Monitoring the Kubernetes cluster and TiDB Operator

This document gives a brief introduction to the two monitoring tasks.

7.1 Monitor the TiDB cluster

You can monitor the TiDB cluster with Prometheus and Grafana. A separate monitoring system is created and configured for each TiDB cluster created by TiDB Operator. The monitoring system runs in the same Namespace as the TiDB cluster, and includes two components - Prometheus and Grafana.

The monitoring data is not persisted by default. If the monitoring container restarts for some reason, the existing monitoring data gets lost. To persist the monitoring data, you can set `monitor.persistent` to `true` in the `values.yaml` file. When you enable this option, you need to set `storageClass` to an existing storage in the current cluster, and this storage is required to support persisting data, otherwise there is still a risk of data loss.

For configuration details on the monitoring system, refer to TiDB Cluster Monitoring.

7.1.1 View the monitoring dashboard

You can run the `kubectl port-forward` command to view the monitoring dashboard:

```
kubectl port-forward -n <namespace> svc/<release-name>-grafana 3000:3000 &>/
    ↪ tmp/portforward-grafana.log &
```

Then open http://localhost:3000 in your browser and log on with the default username and password `admin`.

The Grafana service is exposed by `NodePort` by default. If the Kubernetes cluster supports load balancer, you can change `monitor.grafana.service.type` to `LoadBalancer` in `values.yaml`. Then, after executing `helm upgrade`, access the dashboard through the load balancer.

If there is no need to use Grafana, you can save resources by setting `monitor.grafana.`
`↪ create` to `false` in `values.yaml` during deployment. In this case, you need to use other existing or newly deployed data visualization tools to directly access the monitoring data.

7.1.2 Access the monitoring data

To access the monitoring data directly, run the `kubectl port-forward` command to access Prometheus:

```
kubectl port-forward -n <namespace> svc/<release-name>-prometheus 9090:9090
    ↪ &>/tmp/portforward-prometheus.log &
```

Then open http://localhost:9090 in your browser or access this address via a client tool.

The Prometheus service is exposed by `NodePort` by default. If the Kubernetes cluster supports load balancer, you can change `monitor.prometheus.service.type` to `LoadBalancer` in `values.yaml`. Then, after executing `helm upgrade`, access the monitoring data through the load balancer.

7.2 Monitor the Kubernetes cluster

The TiDB monitoring system deployed with the cluster only focuses on the operation of the TiDB components themselves, and does not include the monitoring of container resources, hosts, Kubernetes components, and TiDB Operator. Monitoring of these components or resources requires the deployment of a monitoring system across the entire Kubernetes cluster dimension.

7.2.1 Monitor the host

Monitoring the host and its resources works in the same way as monitoring physical resources of a traditional server.

If you already have a monitoring system for your physical server in your existing infrastructure, you only need to add the host that holds Kubernetes to the existing monitoring system by conventional means; if there is no monitoring system available, or you want to deploy a separate monitoring system to monitor the host that holds Kubernetes, then you can use any monitoring system that you are familiar with.

The newly deployed monitoring system can run on a separate server, directly on the host that holds Kubernetes, or in a Kubernetes cluster. Different deployment methods might mean differences in the deployment configuration and resource utilization, but there are no major differences in usage.

Some common open source monitoring systems that can be used to monitor server resources are:

- CollectD
- Nagios
- Prometheus & node_exporter
- Zabbix

Some cloud service providers or specialized performance monitoring service providers also have their own free or chargeable monitoring solutions that you can choose from.

It is recommended to deploy a host monitoring system in the Kubernetes cluster via Prometheus Operator based on Node Exporter and Prometheus. This solution can also be compatible with and used for monitoring the Kubernetes' own components.

7.2.2 Monitor Kubernetes components

For monitoring Kubernetes components, you can refer to the solutions provided in the Kubernetes official documentation or use other Kubernetes-compatible monitoring systems.

Some cloud service providers may provide their own solutions for monitoring Kubernetes components, and some specialized performance monitoring service providers have their own Kubernetes integration solutions that you can choose from.

TiDB Operator is actually a container running in Kubernetes. For this reason, you can monitor TiDB Operator by choosing any monitoring system that can monitor the status and resources of a Kubernetes container without deploying additional monitoring components.

It is recommended to deploy a host monitoring system via Prometheus Operator based on Node Exporter and Prometheus. This solution can also be compatible with and used for monitoring host resources.

7.3 Alert configuration

7.3.1 Alerts in the TiDB Cluster

When Prometheus is deployed with a TiDB cluster, some default alert rules are automatically imported. You can view all alert rules and statuses in the current system by accessing the Alerts page of Prometheus through a browser.

Currently, the custom configuration of alert rules is not supported. If you do need to modify the alert rules, you can manually download charts to modify them.

The default Prometheus and alert configuration do not support sending alert messages. To send an alert message, you can integrate Prometheus with any tool that supports Prometheus alerts. It is recommended to manage and send alert messages via AlertManager.

If you already have an available AlertManager service in your existing infrastructure, you can modify `monitor.prometheus.alertmanagerURL` in the `values.yaml` file and configure its address for use by Prometheus; if there is no AlertManager service available, or if you want to deploy a separate set of services, you can refer to Prometheus official document.

7.3.2 Alerts in Kubernetes

If you deploy a monitoring system for Kubernetes hosts and services by Prometheus Operator, some alert rules are configured by default, and an AlertManager service is deployed. For details, see kube-prometheus.

If you monitor Kubernetes hosts and services by using other tools or services, you can consult the corresponding information provided by the tool or service provider.

8 Maintain

8.1 Destroy TiDB Clusters in Kubernetes

This document describes how to deploy TiDB clusters in Kubernetes.

To destroy a TiDB cluster in Kubernetes, run the following commands:

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

The above commands only removes the runining Pod with the data still retained. If you want the data to be deleted as well, you can use the following commands:

> **Warning:**
>
> The following commands deletes your data completely. Please be cautious.

```
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 Restart a TiDB Cluster in Kubernetes

This document describes how to forcibly restart a TiDB cluster in the Kubernetes cluster, including restarting a Pod, restarting all Pods of a component, and restarting all Pods of the TiDB cluster.

> **Note:**
>
> TiDB Operator v1.0.x only supports restarting the Pod in a forcible way.
>
> - If the PD Pod being restarted is the PD leader, its leadership does not automatically transfer, which can cause the PD service to be temporarily unavailable.
> - If the TiKV Pod being restarted contains a Region leader of the TiKV cluster, its leadership does not automatically transfer, which can cause the requests of accessing the corresponding data to fail.
> - Restarting the TiDB Pod can cause the requests of accessing this Pod to fail.

8.2.1 Forcibly restart a Pod

Execute the following command to forcibly restart a Pod:

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

8.2.2 Forcibly restart all Pods of a component

Execute the following command to check the Pod list of a component:

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

Execute the following command to forcibly restart all Pods of a component:

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

Note:

- To forcibly restart all Pods of the `PD`, `TiDB`, or `TiKV` component, replace `<component-name>` in the above commands with `pd`, `tidb`, or `tikv` respectively.
- Replace `<namespace>` in the above commands with your own namespace.

8.2.3 Forcibly restart all Pods of the TiDB cluster

Execute the following command to check the Pod list of the TiDB cluster (including `monitor` and `discovery`):

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

Execute the following command to forcibly restart all Pods of the TiDB cluster (including `monitor` and `discovery`):

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

> **Note:**
>
> Replace `<namespace>` and `<tidb-cluster-name>` in the above commands with your own namespace and TiDB cluster name.

8.3 Maintain Kubernetes Nodes that Hold the TiDB Cluster

TiDB is a highly available database that can run smoothly when some of the database nodes go offline. For this reason, you can safely shut down and maintain the Kubernetes nodes at the bottom layer without influencing TiDB's service. Specifically, you need to adopt various maintenance strategies when handling nodes that hold PD, TiKV, and TiDB instances because of their different features.

This document introduces how to perform a temporary or long-term maintenance task for the Kubernetes nodes.

8.3.1 Prerequisites

- `kubectl`
- `tkctl`
- `jq`

> **Note:**
>
> Before you perform a long-term node maintenance, you need to make sure that the remaining resources in the Kubernetes cluster are enough for running the TiDB cluster.

8.3.2 Maintain nodes that hold PD and TiDB instances

Migrating PD and TiDB instances from a node is relatively fast, so you can proactively evict the instances to other nodes and perform maintenance on the desired node:

1. Check whether there is any TiKV instance on the node to be maintained:

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

   If any TiKV instance is found, see Maintain nodes that hold TiKV instances.

2. Use the `kubectl cordon` command to prevent new Pods from being scheduled to the node to be maintained:

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

3. Use the `kubectl drain` command to migrate the database instances on the maintenance node to other nodes:

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

   After running this command, TiDB instances on this node are automatically migrated to another available node, and the PD instance will trigger the auto-failover mechanism after five minutes and complete the nodes.

4. At this time, if you want to make this Kubernetes node offline, you can delete it by running:

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

   If you want to recover a Kubernetes node, you need to first make sure that it is in a healthy state:

   ```
   watch kubectl get node <node-name>
   ```

   After the node goes into the `Ready` state, you can proceed with the following operations.

5. Use `kubectl uncordon` to lift the scheduling restriction on the node:

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

6. See whether all Pods get back to normal and are running:

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

   Or:

   ```
   watch tkctl get all
   ```

   When you confirm that all Pods are running normally, then you have successfully finished the maintenance task.

8.3.3 Maintain nodes that hold TiKV instances

Migrating TiKV instances is relatively slow and might lead to unwanted data migration load on the cluster. For this reason, you need to choose different strategies as needed prior to maintaining the nodes that hold TiKV instances:

- If you want to maintain a node that is recoverable in a short term, you can recover the TiKV node from where it is after the maintenance without migrating it elsewhere.
- If you want to maintain a node that is not recoverable in a short term, you need to make a plan for the TiKV migration task.

8.3.3.1 Maintain a recoverable node in a short term

For a short-term maintenance, you can increase the TiKV instance downtime that the cluster allows by adjusting the `max-store-down-time` configuration of the PD cluster. You can finish the maintenance and recover the Kubernetes node during this time, and then all TiKV instances on this node will be automatically recovered.

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

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

After configuring `max-store-down-time` to an appropriate value, the follow-up operations are the same as those in Maintain nodes that hold PD and TiDB instances.

8.3.3.2 Maintain an irrecoverable node in a short term

For the maintenance on an node that cannot be recovered in a short term (for example, a node has to go offline for a long time), you need to use `pd-ctl` to proactively tell the TiDB cluster to make the corresponding TiKV instances offline, and manually unbind the instances from the node.

1. Use `kubectl cordon` to prevent new Pods from being scheduled to the node to be maintained:

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

2. Check the TiKV instance on the maintenance node:

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

3. Use `pd-ctl` to proactively put the TiKV instance offline:

> **Note:**
> Before you make the TiKV instances offline, you need to make sure that the number of remaining TiKV instances in the cluster is no smaller than the number of TiKV replicas in the PD configuration (configuration item: `max-replicas`). If this is not the case, you need to scale TiKV first.

   Check `store-id` of the TiKV instance:

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

   Make the TiKV instance offline:

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

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

4. Wait for the store to change its status from `state_name` to `Tombstone`:

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

5. Unbind the TiKV instance from the local drive of the node:

   Get the `PesistentVolumeClaim` used by the Pod:

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

   Delete the `PesistentVolumeClaim`:

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

6. Delete the TiKV instance:

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

7. Check whether the TiKV instance is normally scheduled to another node:

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

   If there are more TiKV instances on the maintenance node, you need to follow the above steps until all instances are migrated to other nodes.

8. After you make sure that there is no more TiKV instance on the node, you can evict other instances on the node:

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

9. Confirm again that there is no more TiKV, TiDB and PD instances running on this node:

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

10. (Optional) If this node is made offline for a long time, it is recommended to delete it from the Kubernetes cluster:

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

Now, you have successfully finished the maintenance task for the node.

8.4 Use PD Recover to Recover the PD Cluster

PD Recover is a disaster recovery tool of PD, used to recover the PD cluster which cannot start or provide services normally.

8.4.1 Download PD Recover

1. Download the official TiDB package:

   ```
   wget https://download.pingcap.org/tidb-${version}-linux-amd64.tar.gz
   ```

   In the command above, `${version}` is the version of the TiDB cluster, such as `v4`
   `↪ .0.0-rc`.

2. Unpack the TiDB package for installation:

   ```
   tar -xzf tidb-${version}-linux-amd64.tar.gz
   ```

   `pd-recover` is in the `tidb-${version}-linux-amd64/bin` directory.

8.4.2 Recover the PD cluster

This section introduces how to recover the PD cluster using PD Recover.

8.4.2.1 Get Cluster ID

```
kubectl get tc ${cluster_name} -n ${namespace} -o='go-template={{.status.
    ↪ clusterID}}{{"\n"}}'
```

Example:

```
kubectl get tc test -n test -o='go-template={{.status.clusterID}}{{"\n"}}'
6821434242797747735
```

8.4.2.2 Get Alloc ID

When you use `pd-recover` to recover the PD cluster, you need to specify `alloc-id`. The value of `alloc-id` must be larger than the largest allocated ID (`Alloc ID`) of the original cluster.

1. Access the Prometheus monitoring data of the TiDB cluster by taking steps in Access the monitoring data.
2. Enter `pd_cluster_id` in the input box and click the `Execute` button to make a query. Get the largest value in the query result.
3. Multiply the largest value in the query result by `100`. Use the multiplied value as the `alloc-id` value specified when using `pd-recover`.

8.4.2.3 Recover the PD Pod

1. Delete the Pod of the PD cluster.

   Execute the following command to set the value of `spec.pd.replicas` to `0`:

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

   Because the PD cluster is in an abnormal state, TiDB Operator cannot synchronize the change above to the PD StatefulSet. You need to execute the following command to set the `spec.replicas` of the PD StatefulSet to `0`.

   ```
   kubectl edit sts ${cluster_name}-pd -n ${namespace}
   ```

   Execute the following command to confirm that the PD Pod is deleted:

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

2. After confirming that all PD Pods are deleted, execute the following command to delete the PVCs bound to the PD Pods:

   ```
   kubectl delete pvc -l app.kubernetes.io/component=pd,app.kubernetes.io/
       ↪ instance=${cluster_name} -n ${namespace}
   ```

3. After the PVCs are deleted, scale out the PD cluster to one Pod:

   Execute the following command to set the value of `spec.pd.replicas` to `1`:

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

   Because the PD cluster is in an abnormal state, TiDB Operator cannot synchronize the change above to the PD StatefulSet. You need to execute the following command to set the `spec.replicas` of the PD StatefulSet to `1`.

   ```
   kubectl edit sts ${cluster_name}-pd -n ${namespace}
   ```

   Execute the following command to confirm that the PD Pod is started:

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

8.4.2.4 Recover the cluster

1. Execute the `port-forward` command to expose the PD service:

   ```
   kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379
   ```

2. Open a **new** terminal tab or window, enter the directory where `pd-recover` is located, and execute the `pd-recover` command to recover the PD cluster:

```
./pd-recover -endpoints http://127.0.0.1:2379 -cluster-id ${cluster_id}
    ↪  -alloc-id ${alloc_id}
```

In the command above, `${cluster_id}` is the cluster ID got in Get Cluster ID. `${ ↪ alloc_id}` is the largest value of `pd_cluster_id` (got in Get Alloc ID) multiplied by `100`.

After the `pd-recover` command is successfully executed, the following result is printed:

```
recover success! please restart the PD cluster
```

3. Go back to the window where the `port-forward` command is executed, press Ctrl+C to stop and exit.

8.4.2.5 Restart the PD Pod

1. Delete the PD Pod:

```
kubectl delete pod ${cluster_name}-pd-0 -n ${namespace}
```

2. After the Pod is started successfully, execute the `port-forward` command to expose the PD service:

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379
```

3. Open a **new** terminal tab or window, execute the following command to confirm the Cluster ID is the one got in Get Cluster ID.

```
curl 127.0.0.1:2379/pd/api/v1/cluster
```

4. Go back to the window where the `port-forward` command is executed, press Ctrl+C to stop and exit.

8.4.2.6 Increase the capacity of the PD cluster

Execute the following command to set the value of `spec.pd.replicas` to the desired number of Pods:

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

8.4.2.7 Restart TiDB and 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 Backup and Restore

8.5.1 Backup and Restore

This document describes how to back up and restore the data of a TiDB cluster in Kubernetes.

TiDB in Kubernetes supports two kinds of backup strategies:

- Full backup (scheduled or ad-hoc): use `mydumper` to take a logical backup of the TiDB cluster.
- Incremental backup: use `TiDB Binlog` to replicate data in the TiDB cluster to another database or take a real-time backup of the data.

Currently, TiDB in Kubernetes only supports automatic restoration for full backup taken by `mydumper`. Restoring the incremental backup data by `TiDB Binlog` requires manual operations.

8.5.1.1 Full backup

Full backup uses `mydumper` to take a logical backup of a TiDB cluster. The backup task creates a PVC (PersistentVolumeClaim) to store data.

In the default configuration, the backup uses PV (Persistent Volume) to store backup data. You can also store the data in Google Cloud Storage buckets, Ceph Object Storage or Amazon S3 by changing the configuration. In this case, the backup data is temporarily stored in the PV before it is uploaded to object storage. Refer to TiDB cluster backup configuration for all configuration options you have.

You can either set up a scheduled full backup job or take a full backup in an ad-hoc manner.

8.5.1.1.1 Scheduled full backup

Scheduled full backup is a task created alongside the TiDB cluster, and it runs periodically like `crontab`.

To configure a scheduled full backup, modify the `scheduledBackup` section in the `values`
↪ `.yaml` file of the TiDB cluster:

1. Set `scheduledBackup.create` to `true`.

2. Set `scheduledBackup.storageClassName` to the `storageClass` of the PV that stores the backup data.

> **Note:**
>
> You must set the scheduled full backup PV's reclaim policy to `Retain` to keep your backup data safe.

3. Configure `scheduledBackup.schedule` in the Cron format to define the scheduling.

4. Create a Kubernetes Secret containing the username and password (the user must have the privileges to back up the data). Meanwhile, set `scheduledBackup.secretName` to the name of the created `Secret`(default to `backup-secret`):

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

5. Create a new TiDB cluster with the scheduled full backup task by running `helm ↪ install`, or enable the scheduled full backup for the existing cluster by `helm ↪ upgrade`:

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

8.5.1.1.2 Ad-hoc full backup

Ad-hoc full backup is encapsulated in a helm chart - `pingcap/tidb-backup`. According to the `mode` configuration in the `values.yaml` file, this chart can perform either full backup or data restoration. The restore section covers how to restore the backup data.

Follow the steps below to perform an ad-hoc full backup task:

1. Modify the `values.yaml` file:

    - Set `clusterName` to the target TiDB cluster name.
    - Set `mode` to `backup`.
    - Set `storage.className` to the `storageClass` of the PV that stores the backup data.
    - Adjust the `storage.size` according to your database size.

> **Note:**
>
> You must set the ad-hoc full backup PV's reclaim policy to `Retain` to keep your backup data safe.

2. Create a Kubernetes Secret containing the username and password (the user must have the privileges to back up the data). Meanwhile, set `secretName` in the `values.yaml` file to the name of the created `Secret`(default to `backup-secret`):

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

3. Run the following command to perform an ad-hoc backup task:

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

8.5.1.1.3 View backups

For backups stored in PV, you can view them by using the following command:

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

If you store your backup data in Google Cloud Storage, Ceph Object Storage or Amazon S3, you can view the backups by using the GUI or CLI tools provided by these storage providers.

8.5.1.2 Restore

The `pingcap/tidb-backup` helm chart helps restore a TiDB cluster using backup data. Follow the steps below to restore:

1. Modify the `values.yaml` file:

    - Set `clusterName` to the target TiDB cluster name.
    - Set `mode` to `restore`.
    - Set `name` to the name of the backup you want to restore (refer to view backups to view available backups). If the backup is stored in Google Cloud Storage, Ceph Object Storage or Amazon S3, you must configure the corresponding sections and make sure that the same configurations are applied as you perform the full backup.

2. Create a Kubernetes Secret containing the user and password (the user must have the privileges to back up the data). Meanwhile, set `secretName` in the `values.yaml` file to the name of the created `Secret` (default to `backup-secret`; skip this if you have already created one when you perform full backup):

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

3. Restore the backup:

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

8.5.1.3 Incremental backup

Incremental backup uses TiDB Binlog to collect binlog data from TiDB and provide near real-time backup and replication to downstream platforms.

For the detailed guide of maintaining TiDB Binlog in Kubernetes, refer to TiDB Binlog.

8.5.1.3.1 Scale in Pump

To scale in Pump, for each Pump node, make the node offline and then run the `helm upgrade` command to delete the corresponding Pump Pod.

1. Make a Pump node offline from the TiDB cluster

   Suppose there are 3 Pump nodes, and you want to get the third node offline and modify `<ordinal-id>` to `2`, run the following command (`<version>` is the current version of TiDB).

   ```
   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
   ```

   Then, check the log output of Pump. If Pump outputs `pump offline, please delete my pod`, the state of the Pump node is successfully switched to `offline`.

   ```
   kubectl logs -f -n <namespace> <release-name>-pump-<ordinal-id>
   ```

2. Delete the corresponding Pump Pod

   Modify `binlog.pump.replicas` in the `values.yaml` file to `2` and then run the following command to delete the Pump Pod.

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

8.5.2 Restore Data into TiDB in Kubernetes

This document describes how to restore data into a TiDB cluster in Kubernetes using TiDB Lightning.

TiDB Lightning contains two components: tidb-lightning and tikv-importer. In Kubernetes, the tikv-importer is inside the Helm chart of the TiDB cluster. And tikv-importer is deployed as a `StatefulSet` with `replicas=1` while tidb-lightning is in a separate Helm chart and deployed as a `Job`.

Therefore, both the tikv-importer and tidb-lightning need to be deployed to restore data with TiDB Lightning.

8.5.2.1 Deploy tikv-importer

The tikv-importer can be enabled for an existing TiDB cluster or for a newly created one.

- Create a new TiDB cluster with tikv-importer enabled

  1. Set `importer.create` to `true` in tidb-cluster `values.yaml`
  2. Deploy the cluster

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

- Configure an existing TiDB cluster to enable tikv-importer

  1. Set `importer.create` to `true` in the `values.yaml` file of the TiDB cluster
  2. Upgrade the existing TiDB cluster

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

8.5.2.2 Deploy tidb-lightning

1. Configure TiDB Lightning

   Use the following command to get the default configuration of TiDB Lightning.

   ```
   helm inspect values pingcap/tidb-lightning --version=<chart-version> >
       ↪ tidb-lightning-values.yaml
   ```

   TiDB Lightning Helm chart supports both local and remote data source.

   - Local

     Local mode requires the Mydumper backup data to be on one of the Kubernetes node. This mode can be enabled by setting `dataSource.local.nodeName` to the node name and `dataSource.local.hostPath` to the Mydumper backup data directory path which contains a file named `metadata`.

   - Remote

     Unlike local mode, remote mode needs to use rclone to download Mydumper backup tarball file from a network storage to a PV. Any cloud storage supported by rclone should work, but currently only the following have been tested: Google Cloud Storage (GCS), AWS S3, Ceph Object Storage.

     1. Make sure that `dataSource.local.nodeName` and `dataSource.local.`
        `↪ hostPath` are commented out.

2. Create a `Secret` containing the rclone configuration. A sample configuration is listed below. Only one cloud storage configuration is required. For other cloud storages, please refer to rclone documentation.

```
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
  # The service account must include Storage Object Viewer role
  # The content can be retrieved by `cat <service-account-file.
      ↪ json> | jq -c .`
  service_account_credentials = <service-account-json-file-
      ↪ content>
```

   Fill in the placeholders with your configurations and save it as `secret.yaml` ↪ , and then create the secret via `kubectl apply -f secret.yaml -n <` ↪ `namespace>`.

3. Configure the `dataSource.remote.storageClassName` to an existing storage class in the Kubernetes cluster.

2. Deploy 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>
```

When TiDB Lightning fails to restore data, it cannot simply be restarted, but manual intervention is required. So the tidb-lightning's `Job` restart policy is set to `Never`.

> **Note:**
>
> Currently, TiDB Lightning will exit with non-zero error code even when data is successfully restored. This will trigger the job failure. Therefore, the success status needs to be determined by viewing tidb-lightning pod's log.

If the lightning fails to restore data, follow the steps below to do manual intervention:

1. Delete the lightning job by running `kubectl delete job -n <namespace> <tidb-lightning-release-name>-tidb-lightning`.
2. Create the lightning job again with `failFast` disabled by `helm template pingcap/tidb-lightning --name <tidb-lightning-release-name> --set failFast=false -f tidb-lightning-values.yaml | kubectl apply -n <namespace> -f -`.
3. When the lightning pod is running again, use `kubectl exec -it -n <namesapce> <tidb-lightning-pod-name> sh` to `exec` into the lightning container.
4. Get the startup script by running `cat /proc/1/cmdline`.
5. Diagnose the lightning following the troubleshooting guide.

8.5.2.3 Destroy TiDB Lightning

Currently, TiDB Lightning can only restore data offline. When the restoration finishes and the TiDB cluster needs to provide service for applications, the TiDB Lightning should be deleted to save cost.

- To delete tikv-importer:
    1. In `values.yaml` of the TiDB cluster chart, set `importer.create` to `false`.
    2. Run `helm upgrade <tidb-cluster-release-name> pingcap/tidb-cluster -f values.yaml`.
- To delete tidb-lightning, run `helm delete <tidb-lightning-release-name> --purge`.

8.6 Collect TiDB Logs in Kubernetes

Runtime logs of the system and program can be very useful for troubleshooting problems and automating some operations. This document introduces the methods to collect logs of TiDB and its related components.

8.6.1 Collect logs of TiDB components in Kubernetes

The TiDB components deployed by TiDB Operator output the logs in the `stdout` and `stderr` of the container by default. For Kubernetes, these logs are stored in the host's `/var/log/containers` directory, and the file name contains information such as the Pod name and the container name. For this reason, you can collect the logs of the application in the container directly on the host.

If you already have a system for collecting logs in your existing infrastructure, you only need to add the `/var/log/containers/*.log` file on the host that holds Kubernetes in the collection scope by conventional means; if there is no available log collection system, or you want to deploy a separate system for collecting relevant logs, you are free to use any system or solution that you are familiar with.

The Kubernetes official documentation provides Stackdriver as a log collection method.

Common open source tools that can be used to collect Kubernetes logs are:

- Fluentd
- Fluent-bit
- Filebeat
- Logstash

Collected Logs can usually be aggregated and stored on a specific server or in a dedicated storage and analysis system such as ElasticSearch.

Some cloud service providers or specialized performance monitoring service providers also have their own free or chargeable log collection options that you can choose from.

If you do not aggregate logs via a separate log collection tool, you can also use the `kubectl` tool directly to view the runtime log of a specific container, but this method does not allow you to view the log of a destroyed container:

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

> **Note:**
>
> To view the log before the container restarts, you can add the `-p` parameter when executing the above command.

If you need to collect logs from multiple Pods, you can use stern:

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

8.6.2 Collect TiDB slow query logs

For versions prior to 3.0, by default, TiDB prints slow query logs to standard output, mixed with application logs.

- For the TiDB version <= 2.1.7, you can filter the slow query logs with the keyword `SLOW_QUERY`, for example:

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

- For the TiDB version >= 2.1.8, it is not so easy to separate the slow query log due to changes to the slow query log format. For this reason, it is recommended to configure `separateSlowLog: true` as described below to view the slow query log separately.

In some cases, you may want to use some tools or automated systems to analyze and process the log content. The application log of each TiDB component uses unified log format, which facilitates parsing with other programs. However, because slow query logs use a multi-line format that is compatible with MySQL, it might be difficult to parse slow query logs when they are mixed with application logs.

If you want to separate the slow query logs from the application logs, you can configure the `separateSlowLog` parameter in the `values.yaml` file. This outputs the slow query log to a dedicated bypass container so that it can be stored in a separate file on the host.

To do this, follow the steps below:

1. Modify the `values.yaml` file and set the `separateSlowLog` parameter to `true`:

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

2. Run `helm upgrade` to apply the configuration.

3. Then you can view the slow query log through the sidecar container named `slowlog`:

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

For 3.0 and the later versions, TiDB outputs slow query logs to a separate `slowlog.log` file, and `separateSlowLog` is enabled by default, so you can view slow query logs directly from the sidecar container without additional settings.

Note:

The format of TiDB slow query logs is the same as that of MySQL slow query logs. However, due to the characteristics of TiDB itself, some of the specific fields may be different. For this reason, the tool for parsing MySQL slow query logs may not be fully compatible with TiDB slow query logs.

8.6.3 Collect system logs

System logs can be collected on Kubernetes hosts in the usual way. If you already have a system for collecting logs in your existing infrastructure, you only need to add the relevant servers and log files in the collection scope by conventional means; if there is no available log collection system, or you want to deploy a separate set of systems for collecting relevant logs, you are free to use any system or solution that you are familiar with.

All of the common log collection tools mentioned above support collecting system logs. Some cloud service providers or specialized performance monitoring service providers also have their own free or chargeable log collection options that you can choose from.

8.7 Automatic Failover

Automatic failover means that when a node in the TiDB cluster fails, TiDB Operator automatically adds a new one to ensure the high availability of the cluster. It works similarly with the `Deployment` behavior in Kubernetes.

TiDB Operator manages Pods based on `StatefulSet`, which does not automatically create a new node to replace the original node when a Pod goes down. For this reason, the automatic failover feature is added to TiDB Operator, which expands the behavior of `StatefulSet`.

The automatic failover feature is enabled by default in TiDB Operator. You can disable it by setting `controllerManager.autoFailover` to `false` in the `charts/tidb-operator/values.yaml` file when deploying TiDB Operator:

```
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
```

By default, `pdFailoverPeriod`, `tikvFailoverPeriod` and `tidbFailoverPeriod` are set to be 5 minutes, which is the waiting timeout after an instance failure is identified. After this time, TiDB Operator begins the automatic failover process.

8.7.1 Automatic failover policies

There are three components in a TiDB cluster - PD, TiKV, and TiDB, each of which has its own automatic failover policy. This section gives an in-depth introduction to these policies.

8.7.1.1 Failover with PD

Assume that there are 3 nodes in a PD cluster. If a PD node is down for over 5 minutes (configurable by modifying `tidbFailoverPeriod`), TiDB Operator takes this node offline first, and creates a new PD node. At this time, there are 4 nodes in the cluster. If the failed PD node gets back online, TiDB Operator deletes the newly created node and the number of nodes gets back to 3.

8.7.1.2 Failover with TiKV

When a TiKV node fails, its status turns to `Disconnected`. After 30 minutes (configurable by modifying `max-store-down-time` in PD's configuration file), it turns to `Down`. After waiting for 5 minutes (configurable by modifying `tikvFailoverPeriod`), TiDB Operator creates a new TiKV node if this TiKV node is still down. If the failed TiKV node gets back online, TiDB Operator does not automatically delete the newly created node, and you need to manually drop it and restore the original number of nodes. To do this, you can delete the TiKV node from the `status.tikv.failureStores` field of the `TidbCluster` object:

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

```
...
status
```

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

After the `cluster1-tikv-0` node turns back to normal, you can delete it as shown below:

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

8.7.1.3 Failover with TiDB

The TiDB automatic failover policy works the same way as `Deployment` does in Kubernetes. Assume that there are 3 nodes in a TiDB cluster. If a TiDB node is down for over 5 minutes (configurable by modifying `tidbFailoverPeriod`), TiDB Operator creates a new TiDB node. At this time, there are 4 nodes in the cluster. When the failed TiDB node gets back online, TiDB Operator deletes the newly created node and the number of nodes gets back to 3.

9 Scale TiDB in Kubernetes

This document introduces how to horizontally and vertically scale a TiDB cluster in Kubernetes.

9.1 Horizontal scaling

Horizontally scaling TiDB means that you scale TiDB out or in by adding or remove nodes in your pool of resources. When you scale a TiDB cluster, PD, TiKV, and TiDB are scaled out or in sequentially according to the values of their replicas. Scaling out operations add nodes based on the node ID in ascending order, while scaling in operations remove nodes based on the node ID in descending order.

9.1.1 Horizontal scaling operations

To perform a horizontal scaling operation:

1. Modify `pd.replicas`, `tidb.replicas`, `tikv.replicas` in the `value.yaml` file of the cluster to a desired value.

2. Run the `helm upgrade` command to scale out or in:

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

3. View the cluster's scaling status:

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

   When the number of Pods for all components reaches the preset value and all components go to the `Running` state, the horizontal scaling is completed.

Note:

- The PD and TiKV components do not trigger scaling in and out operations during the rolling update.
- When the TiKV component scales in, it calls the PD interface to mark the corresponding TiKV instance as offline, and then migrates the data on it to other TiKV nodes. During the data migration, the TiKV Pod is still in the `Running` state, and the corresponding Pod is deleted only after the data migration is completed. The time consumed by scaling in depends on the amount of data on the TiKV instance to be scaled in. You can check whether TiKV is in the `Offline` state by running `kubectl get tidbcluster -n <namespace> <release-name> -↪ o json | jq '.status.tikv.stores'`.
- When the PD and TiKV components scale in, the PVC of the deleted node is retained during the scaling in process. Because the PV's reclaim policy is changed to `Retain`, the data can still be retrieved even if the PVC is deleted.
- The TiKV component does not support scale out while a scale-in operation is in progress. Forcing a scale-out operation might cause anomalies in the cluster. If an anomaly already happens, refer to TiKV Store is in Tombstone status abnormally to fix it.

9.2 Vertical scaling

Vertically scaling TiDB means that you scale TiDB up or down by increasing or decreasing the limit of resources on the node. Vertically scaling is essentially the rolling update of the nodes.

9.2.1 Vertical scaling operations

To perform a vertical scaling operation:

1. Modify `tidb.resources`, `tikv.resources`, `pd.resources` in the `values.yaml` file to a desired value.
2. Run the `helm upgrade` command to upgrade:

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

3. View the progress of the upgrade:

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

When all Pods are rebuilt and in the `Running` state, the vertical scaling is completed.

Note:

- If the resource's `requests` field is modified during the vertical scaling process, because PD and TiKV use `Local PV`, they need to be scheduled back to the original node after the upgrade. At this time, if the original node does not have enough resources, the Pod ends up staying in the `Pending` status and thus impacts the service.
- TiDB is a horizontally scalable database, so it is recommended to take advantage of it simply by adding more nodes rather than upgrading hardware resources like you do with a traditional database.

10 Upgrade

10.1 Perform a Rolling Update to a TiDB Cluster in Kubernetes

When you perform a rolling update to a TiDB cluster in Kubernetes, the Pod is shut down and recreated with the new image or/and configuration serially in the order of PD, TiKV, TiDB. Under the highly available deployment topology (minimum requirements: PD * 3, TiKV * 3, TiDB * 2), performing a rolling update to PD and TiKV servers does not impact the running clients.

- For the clients that can retry stale connections, performing a rolling update to TiDB servers neither impacts the running clients.
- For the clients that **can not** retry stale connections, performing a rolling update to TiDB servers will close the client connections and cause the request to fail. For this situation, it is recommended to add a function for the clients to retry, or to perform a rolling update to TiDB servers in idle time.

10.1.1 Upgrade the version of TiDB cluster

1. Change the `image` of PD, TiKV and TiDB to different image versions in the `values.yaml` file.

2. Run the `helm upgrade` command:

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

3. Check the upgrade progress:

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

10.1.2 Change the configuration of TiDB cluster

By default, changes to the configuration files are applied to the TiDB cluster automatically through a rolling update. You can disable this feature by setting the `enableConfigMapRollout` variable to `false` in the `values.yaml` file, if so, the change of configuration will be loaded until the server being restarted.

You can change the configuration of TiDB cluster through the following steps:

1. Make sure the `enableConfigMapRollout` feature is not disabled explicitly in the `values.yaml` file.

2. Change the configurations in the `values.yaml` file as needed.

3. Run the `helm upgrade` command:

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

4. Check the upgrade process:

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

> **Note:**
>
> Changing the `enableConfigMapRollout` variable against a running cluster will trigger a rolling update of PD, TiKV, TiDB servers even if there's no change to the configuration.

10.1.3 Force an upgrade of TiDB cluster

If the PD cluster is unavailable due to factors such as PD configuration error, PD image tag error and NodeAffinity, then scaling the TiDB cluster, upgrading the TiDB cluster and changing the TiDB cluster configuration cannot be done successfully.

In this case, you can use `force-upgrade` (the version of TiDB Operator must be later than v1.0.0-beta.3) to force an upgrade of the cluster to recover cluster functionality.

First, set `annotation` for the cluster:

```
kubectl annotate --overwrite tc <release-name> -n <namespace> tidb.pingcap.
    ↪ com/force-upgrade=true
```

Then execute the `helm upgrade` command to continue your interrupted operation:

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

> **Warning:**
>
> After the PD cluster recovers, you *must* execute the following command to disable the forced upgrade, or an exception may occur in the next upgrade:
>
> ```
> kubectl annotate tc <release-name> -n <namespace> tidb.pingcap.
>     ↪ com/force-upgrade-
> ```

10.2 Upgrade TiDB Operator and Kubernetes

This document describes how to upgrade TiDB Operator and Kubernetes.

10.2.1 Upgrade TiDB Operator

1. Update CRD (Custom Resource Definition):

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

> **Note:**
> The `<version>` in this document represents the version of TiDB Operator, such as `v1.0.6`. You can check the currently supported version using the `helm search -l tidb-operator` command.

2. Get the `values.yaml` file of the `tidb-operator` chart that you want to install:

```
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. Modify the `operatorImage` image in the `/home/tidb/tidb-operator/<version>/ ↪ values-tidb-operator.yaml` file. Merge the customized configuration in the old `values.yaml` file with the `/home/tidb/tidb-operator/<version>/values-tidb ↪ -operator.yaml` file, and execute `helm upgrade`:

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

10.2.2 Upgrade Kubernetes

When there is a major version upgrade of Kubernetes, you need to make sure that `kubeSchedulerImageTag` matches the version. By default, this value is generated by Helm during the installation or upgrade process. To reset this value, execute `helm upgrade`.

11 Tools

11.1 TiDB Kubernetes Control User Guide

TiDB Kubernetes Control (`tkctl`) is a command line utility that is used for TiDB Operator to maintain and diagnose the TiDB cluster in Kubernetes.

11.1.1 Installation

To install `tkctl`, you can download the pre-built binary or build `tkctl` from source.

11.1.1.1 Download the latest pre-built binary

- MacOS
- Linux
- Windows

After unzipping the downloaded file, you can add the `tkctl` executable file to your `PATH` to finish the installation.

11.1.1.2 Build from source

Requirement: Go >= the 1.11 version or later

```
git clone --depth=1 https://github.com/pingcap/tidb-operator.git && \
GOOS=<YOUR_GOOS> make cli &&\
mv tkctl /usr/local/bin/tkctl
```

11.1.1.3 Shell auto-completion

You can configure the shell auto-completion for `tkctl` to simplify its usage.

To configure the auto-completion for `BASH`, you need to first install the bash-completion package, and configure with either of the two methods below:

- Configure auto-completion in the current shell:

  ```
  source <(tkctl completion bash)
  ```

- Add auto-completion permanently to your bash shell:

  ```
  echo "if hash tkctl 2>/dev/null; then source <(tkctl completion bash);
     ↪ fi" >> ~/.bashrc
  ```

To configure the auto-completion for `ZSH`, you can choose from either of the two methods below:

- Configure auto-completion in the current shell:

  ```
  source <(tkctl completion zsh)
  ```

- Add auto-completion permanently to your zsh shell:

  ```
  echo "if hash tkctl 2>/dev/null; then source <(tkctl completion zsh);
     ↪ fi" >> ~/.zshrc
  ```

11.1.1.4 Kubernetes configuration

`tkctl` reuses the `kubeconfig` file (the default location is `~/.kube/config`) to connect with the Kubernetes cluster. You can verify whether `kubeconfig` is correctly configured by using the following command:

```
tkctl version
```

If the above command correctly outputs the version of TiDB Operator on the server side, then `kubeconfig` is correctly configured.

11.1.2 Commands

11.1.2.1 tkctl version

This command is used to show the version of the local **tkctl** and **tidb-operator** installed in the target cluster.

For example:

```
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
```

11.1.2.2 tkctl list

This command is used to list all installed TiDB clusters.

Flag	Abbreviation	Description
–all-namespaces	-A	Whether to search all Kubernetes namespaces
–output	-o	The output format; you can choose from [default,json,yaml], and the default format is `default`

For example:

```
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
```

11.1.2.3 tkctl use

This command is used to specify the TiDB cluster that the current `tkctl` command operates on. After you specify a TiDB cluster by using this command, all commands that operates on a cluster will automatically select this cluster so the `--tidbcluster` option can be omitted.

For example:

```
tkctl use --namespace=foo demo-cluster
```

```
Tidb cluster switched to foo/demo-cluster
```

11.1.2.4 tkctl info

This command is used to display information about the TiDB cluster.

Flag	Abbreviation	Description
–tidb-cluster	-t	Specify a TiDB cluster; default to the TiDB cluster that is being used

For example:

```
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
```

11.1.2.5 tkctl get [component]

This is a group of commands that are used to get the details of TiDB cluster components.

You can query the following components: `pd`, `tikv`, `tidb`, `volume` and `all` (to query all components).

Flag	Abbreviation	Description
–tidb-cluster	-t	Specify a TiDB cluster; default to the TiDB cluster that is being used
–output	-o	The output format; you can choose from [default,json,yaml], and the default format is `default`

For example:

```
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
```

11.1.2.6 tkctl debug [pod_name]

This command is used to diagnose the Pods in a TiDB cluster. It launches a debug container with the specified docker image on the host that holds the target Pod. The container has the necessary troubleshooting tools installed and shares the namespace with the container in the target Pod, so you can seamlessly diagnose the target container by using various tools in the debug container.

Flag	Abbreviation	Description
– image		Specify the docker image used by the debug container; default to `pingcap/tidb-debug:lastest`
– container	-c	Select the container to be diagnosed; default to the first container of the target Pod

Flag	Abbreviation	Description
–docker-socket		Specify the docker socket on the target node; default to `/var/run/docker.sock`
–privileged		Whether to enable the privileged mode for the debug container

Note:

The default image of the debug container contains various troubleshooting tools, so the image size is relatively large. If you only need `pd-ctl` and `tidb-ctl`, you can specify using the `tidb-control` image by using the `--image=pingcap/tidb-control:latest` command line option.

For example:

```
tkctl debug demo-cluster-tikv-0
```

```
ps -ef
```

Using tools like `GDB` and `perf` in the debug container requires special operations because of the difference in root filesystems of the target container and the debug container.

11.1.2.6.1 GDB

When you use GDB to debug the process in the target container, make sure you set the `program` option to the binary in the **target container**. Additionally, if you use images other than `tidb-debug` as the debug container or if the `pid` of the target process is not 1, you have to configure the location of dynamic libraries via the `set sysroot` command as follows:

```
tkctl debug demo-cluster-tikv-0
```

```
gdb /proc/${pid:-1}/root/tikv-server 1
```

The `.gdbinit` pre-configured in the `tidb-debug` image will set `sysroot` to `/proc/1/`
↪ `root/` automatically. For this reason, you can omit this following step if you are using the `tidb-debug` image and the `pid` of the target process is 1.

```
(gdb) set sysroot /proc/${pid}/root/
```

Start debugging:

```
(gdb) thread apply all bt
```

```
(gdb) info threads
```

11.1.2.6.2 Perf and flame graphs

To use the `perf` command and the `run_flamegraph.sh` script properly, you must copy the program from the target container to the same location in the debug container:

```
tkctl debug demo-cluster-tikv-0
```

```
cp /proc/1/root/tikv-server /
```

```
./run_flamegraph.sh 1
```

This script automatically uploads the generated flame graph (SVG format) to `transfer`
↪ `.sh`, and you can visit the link outputted by the script to download the flame graph.

11.1.2.7 tkctl ctop

The complete form of the command is `tkctl ctop [pod_name | node/node_name ]`.

This command is used to view the real-time monitoring stats of the target Pod or node in the cluster. Compared with `kubectl top`, `tkctl ctop` also provides network and disk stats, which are important for diagnosing problems in the TiDB cluster.

Flag	Abbreviation	Description
–image		Specify the docker image of ctop; default to `quay.io/` ↪ `vektorlab` ↪ `/ctop` ↪ `:0.7.2`

Flag	Abbreviation	Description
--docker-socket		Specify the docker socket that ctop uses; default to `/var/run/docker.sock`

For example:

```
tkctl ctop node/172.16.4.155
```

```
tkctl ctop demo-cluster-tikv-0
```

11.1.2.8 tkctl help [command]

This command is used to print help messages of the sub commands.

For example:

```
tkctl help debug
```

11.1.2.9 tkctl options

This command is used to view the global flags of `tkctl`.

For example:

```
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
```

These options are mainly used to connect with the Kubernetes cluster and two commonly used options among them are as follows:

- `--context`: specify the target Kubernetes cluster
- `--namespace`: specify the Kubernetes namespace

11.2 Tools in Kubernetes

Operations on TiDB in Kubernetes require some open source tools. In the meantime, there are some special requirements for operations using TiDB tools in the Kubernetes environment. This documents introduces in details the related operation tools for TiDB in Kubernetes.

11.2.1 Use PD Control in Kubernetes

PD Control is the command-line tool for PD (Placement Driver). To use PD Control to operate on TiDB clusters in Kubernetes, firstly you need to establish the connection from local to the PD service using `kubectl port-forward`:

```
kubectl port-forward -n <namespace> svc/<cluster-name>-pd 2379:2379 &>/tmp/
    ↪ portforward-pd.log &
```

After the above command is executed, you can access the PD service via `127.0.0.1:2379` ↪ , and then use the default parameters of `pd-ctl` to operate. For example:

```
pd-ctl -d config show
```

Assume that your local port 2379 has been occupied and you want to switch to another port:

```
kubectl port-forward -n <namespace> svc/<cluster-name>-pd <local-port>:2379
    ↪ &>/tmp/portforward-pd.log &
```

Then you need to explicitly assign a PD port for `pd-ctl`:

```
pd-ctl -u 127.0.0.1:<local-port> -d config show
```

11.2.2 Use TiKV Control in Kubernetes

TiKV Control is the command-line tool for TiKV. When using TiKV Control for TiDB clusters in Kubernetes, be aware that each operation mode involves different steps, as described below:

- **Remote Mode**: In this mode, `tikv-ctl` accesses the TiKV service or the PD service through network. Firstly you need to establish the connection from local to the PD service and the target TiKV node using `kubectl port-forward`:

  ```
  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 &
  ```

  After the connection is established, you can access the PD service and the TiKV node via the corresponding port in local:

  ```
  tikv-ctl --host 127.0.0.1:20160 <subcommands>
  ```

  ```
  tikv-ctl --pd 127.0.0.1:2379 compact-cluster
  ```

- **Local Mode:** In this mode, `tikv-ctl` accesses data files of TiKV, and the running TiKV instances must be stopped. To operate in the local mode, first you need to enter the Diagnostic Mode to turn off automatic re-starting for the TiKV instance, stop the TiKV process, and use the `tkctl debug` command to start in the target TiKV Pod a new container that contains the `tikv-ctl` executable. The steps are as follows:

  1. Enter the Diagnostic mode:

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

2. Stop the TiKV process:

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

3. Start the debug container:

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

4. Start using `tikv-ctl` in local mode. It should be noted that the root file system of `tikv` is under `/proc/1/root`, so you need to adjust the path of the data directory accordingly when executing a command:

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

> **Note:**
> The default db path of TiKV instances in the debug container is `/`
> `↪ proc/1/root/var/lib/tikv/db`

11.2.3 Use TiDB Control in Kubernetes

TiDB Control is the command-line tool for TiDB. To use TiDB Control in Kubernetes, you need to access the TiDB node and the PD service from local. It is suggested you turn on the connection from local to the TiDB node and the PD service using `kubectl port-`
`↪ forward`:

```
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 &
```

Then you can use the `tidb-ctl`:

```
tidb-ctl schema in mysql
```

11.2.4 Use Helm

Helm is a package management tool for Kubernetes. Make sure your Helm version >= 2.11.0 and < 2.16.4. The installation steps are as follows:

1. Refer to Helm official documentation to install Helm client.

2. Install Helm server.

   Apply the `RBAC` rule required by the `tiller` component in the cluster and install `tiller`:

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

   If you cannot access `gcr.io`, try using the mirror repository:

   ```
   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]+')
   ```

   Confirm that the tiller pod is in the `running` state by the following command:

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

   If `RBAC` is not enabled for the Kubernetes cluster, use the following command to install `tiller`:

   ```
   helm init --upgrade
   ```

Kubernetes applications are packed as chart in Helm. PingCAP provides the following Helm charts for TiDB in Kubernetes:

- `tidb-operator`: used to deploy TiDB Operator;
- `tidb-cluster`: used to deploy TiDB clusters;
- `tidb-backup`: used to backup or restore TiDB clusters;
- `tidb-lightning`: used to import data into a TiDB cluster;
- `tidb-drainer`: used to deploy TiDB Drainer;
- `tikv-importer`: used to deploy TiKV Importer.

These charts are hosted in the Helm chart repository `https://charts.pingcap.org/` ↪ maintained by PingCAP. You can add this repository to your local using the following command:

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

After adding, use `helm search` to search for the charts provided by PingCAP:

```
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
```

When a new version of chart has been released, you can use `helm repo update` to update the repository cached locally:

```
helm repo update
```

Common Helm operations include `helm install`, `helm upgrade`, and `helm del`. Helm chart usually contains many configurable parameters which could be tedious to configure manually. For convenience, it is recommended that you configure using a YAML file. Based on the conventions in the Helm community, the YAML file used for Helm configuration is named `values.yaml` in this document.

When performing a deployment or upgrade, you must specify the chart name (`chart-name`) and the name for the deployed application (`release-name`). You can also specify one or multiple `values.yaml` files to configure charts. In addition, you can use `chart-version` to specify the chart version (by default the latest GA is used). The steps in command line are as follows:

- Install:

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

- Upgrade (upgrade can be done by modifying the `chart-version` to upgrade to the latest chart version or the `values.yaml` file to update the configuration):

  ```
  helm upgrade <release-name> <chart-name> --version=<chart-version> -f <
      ↪ values-file>
  ```

- To delete the application deployed by Helm, run the following command:

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

For more information on Helm, refer to Helm Documentation.

11.2.5 Use Terraform

Terraform is a Infrastructure as Code management tool. It enables users to define their own infrastructure in a manifestation style, based on which execution plans are generated to create or schedule real world compute resources. TiDB in Kubernetes use Terraform to create and manage TiDB clusters on public clouds.

Follow the steps in Terraform Documentation to install Terraform.

12 Components

12.1 TiDB Scheduler

TiDB Scheduler is a TiDB implementation of Kubernetes scheduler extender. TiDB Scheduler is used to add new scheduling rules to Kubernetes. This document introduces these new scheduling rules and how TiDB Scheduler works.

12.1.1 TiDB cluster scheduling requirements

A TiDB cluster includes three key components: PD, TiKV, and TiDB. Each consists of multiple nodes: PD is a Raft cluster, and TiKV is a multi-Raft group cluster. PD and TiKV components are stateful. The default scheduling rules of the Kubernetes scheduler cannot meet the high availability scheduling requirements of the TiDB cluster, so the Kubernetes scheduling rules need to be extended.

TiDB Scheduler implements the following customized scheduling rules:

12.1.1.1 PD component

Scheduling rule 1: Make sure that the number of PD instances scheduled on each node is less than `Replicas / 2`. For example:

PD cluster size (Replicas)	Maximum number of PD instances that can be scheduled on each node
1	1
2	1
3	1
4	1
5	2
…	

12.1.1.2 TiKV component

Scheduling rule 2: If the number of Kubernetes nodes is less than three (in this case, TiKV cannot achieve high availability), scheduling is not limited; otherwise, the number of TiKV instances that can be scheduled on each node is no more than `ceil(Replicas / 3)`. For example:

TiKV cluster size (Replicas)	Maximum number of TiKV instances that can be scheduled on each node	Best scheduling distribution
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 component

Scheduling rule 3: When you perform a rolling update to a TiDB instance, the instance tends to be scheduled back to its original node.

This ensures stable scheduling and is helpful for the scenario of manually configuring Node IP and NodePort to the LB backend. It can reduce the impact on the cluster during the rolling update because you do not need to adjust the LB configuration when the Node IP is changed after the upgrade.

12.1.2 How TiDB Scheduler works

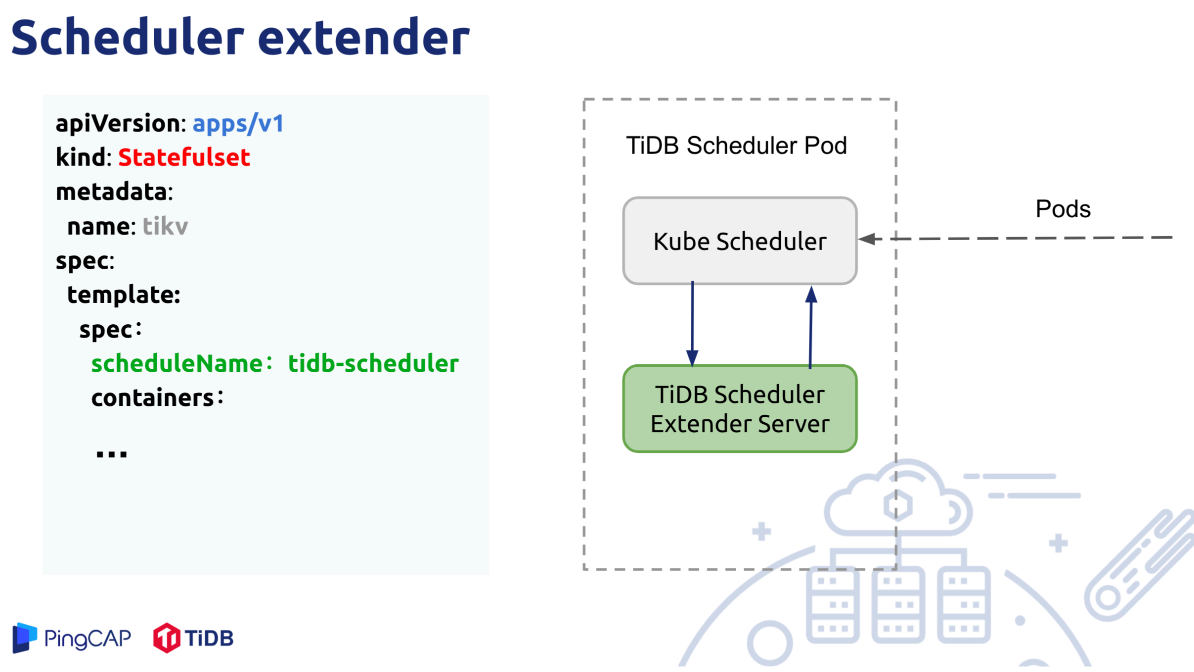


Figure 13: TiDB Scheduler Overview

TiDB Scheduler adds customized scheduling rules by implementing Kubernetes Scheduler extender.

The TiDB Scheduler component is deployed as one or more Pods, though only one Pod is working at the same time. Each Pod has two Containers inside: one Container is a native `kube-scheduler`, and the other is a `tidb-scheduler` implemented as a Kubernetes scheduler extender.

The `.spec.schedulerName` attribute of PD, TiDB, and TiKV Pods created by the TiDB Operator is set to `tidb-scheduler`. This means that the TiDB Scheduler is used for the scheduling.

If you are using a testing cluster and do not require high availability, you can change `.spec.schedulerName` into `default-scheduler` to use the built-in Kubernetes scheduler.

The scheduling process of a Pod is as follows:

- First, `kube-scheduler` pulls all Pods whose `.spec.schedulerName` is `tidb-scheduler`. And Each Pod is filtered using the default Kubernetes scheduling rules.
- Then, `kube-scheduler` sends a request to the `tidb-scheduler` service. Then `tidb-scheduler` filters the sent nodes through the customized scheduling rules (as mentioned above), and returns schedulable nodes to `kube-scheduler`.

- Finally, `kube-scheduler` determines the nodes to be scheduled.

If a Pod cannot be scheduled, see the troubleshooting document to diagnose and solve the issue.

13 Troubleshoot TiDB in Kubernetes

This document describes some common issues and solutions when you use a TiDB cluster in Kubernetes.

13.1 Use the diagnostic mode

When a Pod is in the `CrashLoopBackoff` state, the containers in the Pod quit continually. As a result, you cannot use `kubectl exec` or `tkctl debug` normally, making it inconvenient to diagnose issues.

To solve this problem, TiDB in Kubernetes provides the Pod diagnostic mode for PD, TiKV, and TiDB components. In this mode, the containers in the Pod hang directly after starting, and will not get into a state of repeated crash. Then you can use `kubectl exec` or `tkctl debug` to connect to the Pod containers for diagnosis.

To use the diagnostic mode for troubleshooting:

1. Add an annotation to the Pod to be diagnosed:

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

   The next time the container in the Pod is restarted, it detects this annotation and enters the diagnostic mode.

2. Wait for the Pod to enter the Running state.

   ```
   watch kubectl get pod <pod-name> -n <namespace>
   ```

3. Start the diagnosis.

   Here's an example of using `kubectl exec` to get into the container for diagnosis:

   ```
   kubectl exec -it <pod-name> -n <namespace> -- /bin/sh
   ```

4. After finishing the diagnosis and resolving the problem, delete the Pod.

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

   After the Pod is rebuilt, it automatically returns to the normal mode.

13.2 Recover the cluster after accidental deletion

TiDB Operator uses PV (Persistent Volume) and PVC (Persistent Volume Claim) to store persistent data. If you accidentally delete a cluster using `helm delete`, the PV/PVC objects and data are still retained to ensure data safety.

To restore the cluster at this time, use the `helm install` command to create a cluster with the same name. The retained PV/PVC and data are reused.

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

13.3 Pod is not created normally

After creating a cluster using `helm install`, if the Pod is not created, you can diagnose it using the following commands:

```
kubectl get tidbclusters -n <namespace>
kubectl get statefulsets -n <namespace>
kubectl describe statefulsets -n <namespace> <release-name>-pd
```

13.4 Network connection failure between Pods

In a TiDB cluster, you can access most Pods by using the Pod's domain name (allocated by the Headless Service). The exception is when TiDB Operator collects the cluster information or issues control commands, it accesses the PD (Placement Driver) cluster using the `service-name` of the PD service.

When you find some network connection issues between Pods from the log or monitoring metrics, or you find the network connection between Pods might be abnormal according to the problematic condition, you can follow the following process to diagnose and narrow down the problem:

1. Confirm that the endpoints of the Service and Headless Service are normal:

   ```
   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
   ```

   The `ENDPOINTS` field shown in the above command should be a comma-separated list of `cluster_ip:port`. If the field is empty or incorrect, check the health of the Pod and whether `kube-controller-manager` is working properly.

2. Enter the Pod's Network Namespace to diagnose network problems:

```
tkctl debug -n <namespace> <pod-name>
```

After the remote shell is started, use the `dig` command to diagnose the DNS resolution. If the DNS resolution is abnormal, refer to Debugging DNS Resolution for troubleshooting.

```
dig <HOSTNAME>
```

Use the `ping` command to diagnose the connection with the destination IP (the ClusterIP resolved using `dig`):

```
ping <TARGET_IP>
```

- If the `ping` check fails, refer to Debugging Kubernetes Networking for troubleshooting.
- If the `ping` check succeeds, continue to check whether the target port is open by using `telnet`:

  ```
  telnet <target_ip> <target_port>
  ```

  If the `telnet` check fails, check whether the port corresponding to the Pod is correctly exposed and whether the applied port is correctly configured:

  ```
  # Checks whether the ports are consistent.
  kubectl -n <namespace> get po <pod-name> -ojson | jq '.spec.
      ↪ containers[].ports[].containerPort'

  # Checks whether the application is correctly configured to serve
      ↪ the specified port.
  # The default port of PD is 2379 when not configured.
  kubectl -n <namespace> -it exec <pod-name> -- cat /etc/pd/pd.toml |
      ↪  grep client-urls
  # The default port of PD is 20160 when not configured.
  kubectl -n <namespace> -it exec <pod-name> -- cat /etc/tikv/tikv.
      ↪ toml | grep addr
  # The default port of TiDB is 4000 when not configured.
  kubectl -n <namespace> -it exec <pod-name> -- cat /etc/tidb/tidb.
      ↪ toml | grep port
  ```

13.5 The Pod is in the Pending state

The Pending state of a Pod is usually caused by conditions of insufficient resources, such as:

- The StorageClass of the PVC used by PD, TiKV, Monitor Pod does not exist or the PV is insufficient.
- No nodes in the Kubernetes cluster can satisfy the CPU or memory resources requested by the Pod
- The number of TiKV or PD replicas and the number of nodes in the cluster do not satisfy the high availability scheduling policy of tidb-scheduler

You can check the specific reason for Pending by using the `kubectl describe pod` command:

```
kubectl describe po -n <namespace> <pod-name>
```

- If the CPU or memory resources are insufficient, you can lower the CPU or memory resources requested by the corresponding component for scheduling, or add a new Kubernetes node.

- If the `StorageClass` of the PVC cannot be found, change `storageClassName` in the `values.yaml` file to the name of the `StorageClass` available in the cluster; run `helm ↪ upgrade`; and delete Statefulset and the corresponding PVCs. Run the following command to get the `StorageClass` available in the cluster:

  ```
  kubectl get storageclass
  ```

- If a `StorageClass` exists in the cluster but the available PV is insufficient, you need to add PV resources correspondingly. For Local PV, you can expand it by referring to Local PV Configuration.

- tidb-scheduler has a high availability scheduling policy for TiKV and PD. For the same TiDB cluster, if there are N replicas of TiKV or PD, then the number of PD Pods that can be scheduled to each node is `M=(N-1)/2` (if N<3, then M=1) at most, and the number of TiKV Pods that can be scheduled to each node is `M=ceil(N/3)` (if N<3, then M=1; `ceil` means rounding up) at most. If the Pod's state becomes `Pending` because the high availability scheduling policy is not satisfied, you need to add more nodes in the cluster.

13.6 The Pod is in the `CrashLoopBackOff` state

A Pod in the `CrashLoopBackOff` state means that the container in the Pod repeatedly aborts, in the loop of abort - restart by `kubelet` - abort. There are many potential causes of `CrashLoopBackOff`. In this case, the most effective way to locate it is to view the log of the Pod container:

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

If the log fails to help diagnose the problem, you can add the `-p` parameter to output the log information when the container was last started:

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

After checking the error messages in the log, you can refer to Cannot start `tidb-server`, Cannot start `tikv-server`, and Cannot start `pd-server` for further troubleshooting.

When the "cluster id mismatch" message appears in the TiKV Pod log, it means that the TiKV Pod might have used old data from other or previous TiKV Pod. If the data on the local disk remain uncleared when you configure local storage in the cluster, or the data is not recycled by the local volume provisioner due to a forced deletion of PV, an error might occur.

If you are confirmed that the TiKV should join the cluster as a new node and that the data on the PV should be deleted, you can delete the TiKV Pod and the corresponding PVC. The TiKV Pod automatically rebuilds and binds the new PV for use. When configuring local storage, delete local storage on the machine to avoid Kubernetes using old data. In cluster operation and maintenance, manage PV using the local volume provisioner and do not delete it forcibly. You can manage the lifecycle of PV by creating, deleting PVCs, and setting `reclaimPolicy` for the PV.

In addition, TiKV might also fail to start when `ulimit` is insufficient. In this case, you can modify the `/etc/security/limits.conf` file of the Kubernetes node to increase the `ulimit`:

```
root soft nofile 1000000
root hard nofile 1000000
root soft core unlimited
root soft stack 10240
```

If you cannot confirm the cause from the log and `ulimit` is also a normal value, troubleshoot it further by using the diagnostic mode.

13.7 Unable to access the TiDB service

If you cannot access the TiDB service, first check whether the TiDB service is deployed successfully using the following method:

1. Check whether all components of the cluster are up and the status of each component is `Running`.

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

2. Check the log of TiDB components to see whether errors are reported.

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

If the cluster is successfully deployed, check the network using the following steps:

1. If you cannot access the TiDB service using `NodePort`, try to access the TiDB service using the service domain or `clusterIP` on the node. If the `serviceName` or `clusterIP` works, the network within the Kubernetes cluster is normal. Then the possible issues are as follows:

    - Network failure exists between the client and the node.
    - Check whether the `externalTrafficPolicy` attribute of the TiDB service is `Local`. If it is `Local`, you must access the client using the IP of the node where the TiDB Pod is located.

2. If you still cannot access the TiDB service using the service domain or `clusterIP`, connect using `<PodIP>:4000` on the TiDB service backend. If the `PodIP` works, you can confirm that the problem is in the connection between the service domain and `PodIP` or between `clusterIP` and `PodIP`. Check the following items:

    - Check whether the DNS service works well.

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

    - Check whether `kube-proxy` on each node is working.

    ```
    kubectl get po -n kube-system -l k8s-app=kube-proxy
    ```

    - Check whether the TiDB service rule is correct in the `iptables` rules.

    ```
    iptables-save -t nat |grep <clusterIP>
    ```

    - Check whether the corresponding endpoint is correct.

3. If you cannot access the TiDB service even using `PodIP`, the problem is on the Pod level network. Check the following items:

    - Check whether the relevant route rules on the node are correct.
    - Check whether the network plugin service works well.
    - Refer to network connection failure between Pods section.

13.8 TiKV Store is in `Tombstone` status abnormally

Normally, when a TiKV Pod is in a healthy state (`Running`), the corresponding TiKV store is also in a healthy state (`UP`). However, concurrent scale-in or scale-out on TiKV components might cause part of TiKV stores to fall into the `Tombstone` state abnormally. When this happens, try the following steps to fix it:

1. View the state of the TiKV store:

```
kubectl get -n <namespace> tidbcluster <release-name> -ojson | jq '.
    ↪ status.tikv.stores'
```

2. View the state of the TiKV Pod:

```
kubectl get -n <namespace> po -l app.kubernetes.io/component=tikv
```

3. Compare the state of the TiKV store with that of the Pod. If the store corresponding to a TiKV Pod is in the `Offline` state, it means the store is being taken offline abnormally. You can use the following commands to cancel the offline process and perform recovery operations:

   1. Open the connection to the PD service:

   ```
   kubectl port-forward -n <namespace> svc/<cluster-name>-pd <local-
       ↪ port>:2379 &>/tmp/portforward-pd.log &
   ```

   2. Bring online the corresponding store:

   ```
   curl -X POST http://127.0.0.1:2379/pd/api/v1/store/<store-id>/state
       ↪ ?state=Up
   ```

4. If the TiKV store with the latest `lastHeartbeatTime` that corresponds to a Pod is in a `Tombstone` state, it means that the offline process is completed. At this time, you need to re-create the Pod and bind it with a new PV to perform recovery by taking the following steps:

   1. Set the `reclaimPolicy` value of the PV corresponding to the store to `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. Remove the PVC used by the Pod:

   ```
   kubectl delete -n <namespace> pvc tikv-<pod-name> --wait=false
   ```

   3. Remove the Pod, and wait for it to be re-created:

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

   After the Pod is re-created, a new store is registered in the TiKV cluster. Then the recovery is completed.

13.9 Long queries are abnormally interrupted in TiDB

Load balancers often set the idle connection timeout. If no data is sent over a connection for a specific period of time, load balancer closes the connection.

If a long query is interrupted when you use TiDB, check the middleware program between the client and the TiDB server. If the idle timeout is not long enough for your query, try to set the timeout to a larger value. If you cannot reset it, enable the `tcp-keep-alive` option in TiDB.

In Linux, the keepalive probe packet is sent every 7,200 seconds by default. To shorten the interval, configure `sysctls` via the `podSecurityContext` field.

- If `--allowed-unsafe-sysctls=net.*` can be configured for kubelet in the Kubernetes cluster, configure this parameter for kubelet and configure TiDB in the following way:

```
tidb:
  ...
  podSecurityContext:
    sysctls:
    - name: net.ipv4.tcp_keepalive_time
      value: "300"
```

- If `--allowed-unsafe-sysctls=net.*` cannot be configured for kubelet, configure TiDB in the following way:

```
tidb:
  annotations:
    tidb.pingcap.com/sysctl-init: "true"
  podSecurityContext:
    sysctls:
    - name: net.ipv4.tcp_keepalive_time
      value: "300"
  ...
```

Note:

The configuration above requires TiDB Operator 1.1 or later version.

14 TiDB FAQs in Kubernetes

This document collects frequently asked questions (FAQs) about the TiDB cluster in Kubernetes.

14.1 How to modify time zone settings?

The default time zone setting for each component container of a TiDB cluster in Kubernetes is UTC. To modify this setting, take the steps below based on your cluster status:

- If it is the first time you deploy the cluster:

  In the `values.yaml` file of the TiDB cluster, modify the `timezone` setting. For example, you can set it to `timezone: Asia/Shanghai` before you deploy the TiDB cluster.

- If the cluster is running:

  - In the `values.yaml` file of the TiDB cluster, modify `timezone` settings in the `values.yaml` file of the TiDB cluster. For example, you can set it to `timezone:` ↪ `Asia/Shanghai` and then upgrade the TiDB cluster.
  - Refer to Time Zone Support to modify TiDB service time zone settings.

14.2 Can HPA or VPA be configured on TiDB components?

Currently, the TiDB cluster does not support HPA (Horizontal Pod Autoscaling) or VPA (Vertical Pod Autoscaling), because it is difficult to achieve autoscaling on stateful applications such as a database. Autoscaling can not be achieved merely by the monitoring data of CPU and memory.

14.3 What scenarios require manual intervention when I use TiDB Operator to orchestrate a TiDB cluster?

Besides the operation of the Kubernetes cluster itself, there are the following two scenarios that might require manual intervention when using TiDB Operator:

- Adjusting the cluster after the auto-failover of TiKV. See Auto-Failover for details;
- Maintaining or dropping the specified Kubernetes nodes. See Maintaining Nodes for details.

14.4 What is the recommended deployment topology when I use TiDB Operator to orchestrate a TiDB cluster on a public cloud?

To achieve high availability and data safety, it is recommended that you deploy the TiDB cluster in at least three availability zones in a production environment.

In terms of the deployment topology relationship between the TiDB cluster and TiDB services, TiDB Operator supports the following three deployment modes. Each mode has its own merits and demerits, so your choice must be based on actual application needs.

- Deploy the TiDB cluster and TiDB services in the same Kubernetes cluster of the same VPC;
- Deploy the TiDB cluster and TiDB services in different Kubernetes clusters of the same VPC;
- Deploy the TiDB cluster and TiDB services in different Kubernetes clusters of different VPCs.

14.5 Does TiDB Operator support TiSpark?

TiDB Operator does not yet support automatically orchestrating TiSpark.

If you want to add the TiSpark component to TiDB in Kubernetes, you must maintain Spark on your own in **the same** Kubernetes cluster. You must ensure that Spark can access the IPs and ports of PD and TiKV instances, and install the TiSpark plugin for Spark. TiSpark offers a detailed guide for you to install the TiSpark plugin.

To maintain Spark in Kubernetes, refer to Spark on Kubernetes.

14.6 How to check the configuration of the TiDB cluster?

To check the configuration of the PD, TiKV, and TiDB components of the current cluster, run the following command:

- Check the PD configuration file:

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

- Check the TiKV configuration file:

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

- Check the TiDB configuration file:

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

14.7 Why does TiDB Operator fail to schedule Pods when I deploy the TiDB clusters?

Three possible reasons:

- Insufficient resource or HA Policy causes the Pod stuck in the `Pending` state. Refer to Troubleshoot TiDB in Kubernetes for more details.

- `taint` is applied to some nodes, which prevents the Pod from being scheduled to these nodes unless the Pod has the matching `toleration`. Refer to taint & toleration for more details.

- Scheduling conflict, which causes the Pod stuck in the `ContainerCreating` state. In such case, you can check if there is more than one TiDB Operator deployed in the Kubernetes cluster. Conflicts occur when custom schedulers in multiple TiDB Operators schedule the same Pod in different phases.

  You can execute the following command to verify whether there is more than one TiDB Operator deployed. If more than one record is returned, delete the extra TiDB Operator to resolve the scheduling conflict.

  ```
  kubectl get deployment --all-namespaces |grep tidb-scheduler
  ```

15 Release Notes

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
