TiDB in Kubernetes Documentation

PingCAP Inc.

20230330

Table of Contents

1 TiDB in Kubernetes Docs

2 Introduction

2.1 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.

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)

2.1.1 Manage TiDB clusters using TiDB Operator

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

- For test environment:
  - Get Started using kind, Minikube, or the Google Cloud Shell
- 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
- Back up and Restore a TiDB Cluster
- Automatic Failover
- Monitor a TiDB Cluster in Kubernetes
- View 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.

2.2 What's New in TiDB Operator 1.1

Based on 1.0, TiDB Operator 1.1 has several new features, including TiDB 4.0 support, TiKV data encryption, and TLS certificate configuration. TiDB Operator v1.1 also supports deploying new components such as TiFlash and TiCDC.

TiDB Operator 1.1 also makes improvements in usability, providing the user experience that is consistent with the Kubernetes native resources.

2.2.1 Extensibility

- `TidbCluster` CR supports deploying and managing the PD Discovery component, which is fully capable of replacing tidb-cluster chart to manage the TiDB cluster.
- `TidbCluster` CR adds support for Pump, TiFlash, TiCDC, and TiDB Dashboard.
- Add the Admission Controller (optional) to improve the user experience of upgrade and scaling, and to provide the canary release feature.
- `tidb-scheduler` supports high availability (HA) scheduling at any dimension and scheduler preemption.
- Support using tikv-importer chart to deploy and manage tikv-importer.

2.2.2 Usability

- Add `TidbMonitor` CR to deploy the cluster monitoring.
- Add `TidbInitializer` CR to initialize the cluster.
- Add `Backup`, `BackupSchedule`, and `Restore` CR to back up and restore the cluster, which supports using Amazon S3 or GCS as the remote storage.
- Support gracefully restart a component in the TiDB cluster.

2.2.3 Security

- Support configuring TLS certificates for the TiDB cluster components and for MySQL clients.
- Support TiKV data encryption.

2.2.4 Experimental features

- Add `TidbClusterAutoScaler` to implement auto-scaling. You can enable this feature by turning on the `AutoScaling` switch.
- Add the optional Advanced StatefulSet Controller, which supports deleting a specific Pod. You can enable this feature by turning on the `AdvancedStatefulSet` switch.

For the full release notes, see 1.1 CHANGE LOG.

To deploy TiDB Operator in Kubernetes, see Deployment. For CRD documentation, see API references.

2.3 TiDB Operator v1.1 Notes

This document introduces important notes for TiDB Operator v1.1.

2.3.1 PingCAP no longer updates or maintains tidb-cluster chart

Since TiDB Operator v1.1.0, PingCAP no longer updates the tidb-cluster chart. The components and features that have been managed using the tidb-cluster chart will be managed in new ways in v1.1. See the following table for details:

Components/Features	Managements in TiDB Operator v1.1
TiDB Cluster (PD, TiDB, TiKV)	TidbCluster CR
TiDB Monitor	TidbMonitor CR
TiDB Initializer	TidbInitializer CR
Scheduled Backup	BackupSchedule CR
Pump	TidbCluster CR
Drainer	tidb-drainer chart
Importer	tikv-importer chart

- PingCAP will continue releasing new versions of the tidb-cluster chart but will not add new features to it. However, community members can still add new features to it.
- If you have deployed your TiDB cluster by TiDB Operator (v1.0.0 <= version < v1.1), after TiDB Operator is upgraded to v1.1, you can still upgrade and manage your TiDB cluster using the tidb-cluster chart v1.1.

2.3.2 Switch the components and features managed by tidb-cluster chart to services fully supported by TiDB Operator v1.1

In TiDB Operator v1.1, you can still manage your cluster using Helm and the tidb-cluster chart. Although new features will not be added to the tidb-cluster chart, you can contribute new features to the tidb-cluster chart by yourself, or switch to services that are fully supported by TiDB Operator v1.1.

This section describes how to switch your components and features to services in TiBD Operator v1.1.

2.3.2.1 Discovery

The Discovery service is generated by TiDB Operator automatically, so you do not need to configure it on your own.

2.3.2.2 PD/TiDB/TiKV

In tidb-cluster chart, the configurations of PD, TiDB, and TiKV are rendered into ConfigMap by Helm.

Since TiDB Operator v1.1, you can configure TiDB, TiKV, and PD directly in TidbCluster Custom Resource (CR). For configuration details, refer to Configure a TiDB Cluster using TidbCluster.

> **Note:**
>
> The way TiDB Operator renders configurations is different from that of Helm. If you migrate configurations from `tidb-cluster chart values.yaml` to CR, the corresponding components might be rolling updated.

2.3.2.3 Monitor

To create the TidbMonitor CR and manage the Monitor component, refer to Monitor a TiDB Cluster.

> **Note:**
>
> - The `metadata.name` in the TidbMonitor CR must be consistent with the name of the TidbCluster CR in the cluster.
> - The way TiDB Operator renders resources is different from that of Helm. If you migrate configurations from `tidb-cluster chart values.yaml` to TidbMonitor CR, the Monitor component might be rolling updated.

2.3.2.4 Initializer

- If the initialization job is executed before TiDB Operator is upgraded to v1.1, the initialization job does not need to be migrated from the tidb-cluster chart to the TidbInitializer CR.
- If the initialization job is not executed before TiDB Operator is upgraded to v1.1, and the password for the TiDB root user is not modified, you need to initialize your cluster after upgrading to TiDB Operator v1.1. For details, refer to Initialize a TiDB Cluster in Kubernetes.

2.3.2.5 Pump

After TiDB Operator is upgraded to v1.1, you can modify the TidbCluster CR and add the configuration of Pump. You can then manage the Pump component using TidbCluster CR.

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

You can modify `version`, `replicas`, `storageClassName`, `requests.storage`, and other configurations according to the needs of your cluster.

> **Note:**
>
> The way TiDB Operator renders resources is different from that of Helm. If you migrate configurations from `tidb-cluster chart values.yaml` to TidbCluster CR, the Pump component might be rolling updated.

2.3.2.6 Scheduled Backup

After TiDB Operator is upgraded to v1.1, you can configure the scheduled full backup using BackupSchedule CR:

- If the TiDB cluster version < v3.1, refer to Scheduled backup using Dumpling
- If the TiDB cluster version >= v3.1, refer to Scheduled backup using BR

> **Note:**

- Currently, with BackupSchedule CR, you can back up data only to S3 and GCS using Dumpling, and back up data to S3 using BR. If you perform the scheduled full backup and send data to local Persistent Volume Claim (PVC) before the upgrade, you cannot switch to the CR management after the upgrade.
- If you switch to the CR management, to avoid repeated backup, delete the Cronjob of the previous scheduled full backup.

2.3.2.7 Drainer

- If Drainer is not deployed before TiDB Operator is upgraded to v1.1, you can deploy Drainer as in Deploy multiple drainers.
- If Drainer is already deployed using the tidb-drainer chart before TiDB Operator is upgraded to v1.1, it is recommended to continue managing Drainer using the tidb-drainer chart.
- If Drainer is already deployed using the tidb-cluster chart before TiDB Operator is upgraded to v1.1, it is recommended to manage Drainer using kubectl.

2.3.2.8 TiKV Importer

- If TiKV Importer is not deployed before TiDB Operator is upgraded to v1.1, you can deploy TiKV Importer as in Deploy TiKV Importer.
- If TiKV Importer is already deployed before TiDB Operator is upgraded to v1.1, it is recommended to manage TiKV Importer using kubectl.

2.3.3 Switch other components or features managed by chart to services supported by TiDB Operator v1.1

This section describes how to switch other components and features managed by the chart to services in TiBD Operator v1.1.

2.3.4 Ad-hoc backup

After TiDB Operator is upgraded to v1.1, you can perform backup using the Backup CR. Backup with Dumpling supports full backup, and backup with BR supports both full backup and incremental backup.

- If the TiDB cluster version < v3.1, refer to Ad-hoc full backup using Dumpling.
- If the TiDB cluster version >= v3.1, refer to Ad-hoc backup using BR.

Note:

Currently, with Backup CR, you can back up data only to S3 and GCS using Dumpling or BR. If you perform the scheduled full backup and send data to local Persistent Volume Claim (PVC) before the upgrade, you cannot switch to the CR management after the upgrade.

2.3.5 Restoration

After the TiDB Operator is upgraded to v1.1, you can restore data using the Restore CR.

- If the TiDB cluster version < v3.1, refer to Restore Data from S3-Compatible Storage Using TiDB Lightning.
- If the TiDB cluster version >= v3.1, refer to Restore Data from S3-Compatible Storage Using BR.

Note:

Currently, with Restore CR, you can use TiDB Lightning or BR to restore data from S3 and GCS. If you need to restore the backup data from local Persistent Volume Claim (PVC), you cannot switch to the CR management.

3 Get Started with TiDB Operator in Kubernetes

This document explains how to create a simple Kubernetes cluster and use it to do a basic test deployment of a TiDB cluster using TiDB Operator.

Warning:

This document is for demonstration purposes only. **Do not** follow it in production environments. For production environments, see the instructions in **Deploy** > **Deploy TiDB Cluster**.

These are the steps this document follows:

1. Create a Kubernetes test cluster
2. Deploy TiDB Operator
3. Deploy a TiDB cluster and its monitoring services
4. Connect to a TiDB cluster
5. Upgrade a TiDB cluster
6. Destroy a TiDB cluster

If you have already created a Kubernetes cluster, you can skip to step 2, Deploy TiDB Operator.

If you want to do a production-grade deployment, refer to one of these resources:

- On public cloud:
  - Deploy TiDB on AWS EKS
  - Deploy TiDB on GCP GKE (beta)
  - Deploy TiDB on Alibaba Cloud ACK
- In a self-managed Kubernetes cluster:
  - Familiarize yourself with Prerequisites for TiDB in Kubernetes
  - Configure the local PV for your Kubernetes cluster to achieve high performance for TiKV
  - Deploy TiDB Operator in Kubernetes
  - Deploy TiDB in General Kubernetes

3.1 Create a Kubernetes test cluster

This section covers 2 different ways to create a simple Kubernetes cluster that can be used to test a TiDB cluster running under TiDB Operator. Choose whichever best matches your environment or experience level.

- Using kind (Kubernetes in Docker)
- Using minikube (Kubernetes running locally in a VM)

You can alternatively deploy a Kubernetes cluster in Google Kubernetes Engine in Google Cloud Platform using the Google Cloud Shell, and follow an integrated tutorial to deploy TiDB Operator and the TiDB cluster:

- Open in Google Cloud Shell

3.1.1 Create a Kubernetes cluster using kind

This section shows how to deploy a Kubernetes cluster using kind.

kind is a tool for running local Kubernetes clusters using Docker containers as cluster nodes. It is developed for testing local Kubernetes clusters. The Kubernetes cluster version depends on the node image that kind uses, and you can specify the image to be used for the nodes and choose any other published version. Refer to Docker Hub to see available tags.

> **Warning:**
>
> The kind cluster is for test purposes only. **Do not use** in production.

Before deployment, make sure the following requirements are satisfied:

- Docker: version >= 17.03
- kubectl: version >= 1.12
- kind: version >= 0.8.0
- If you use Linux, the value of the sysctl parameter net.ipv4.ip_forward should be set to 1

The following is an example of using `kind` v0.8.1:

```
kind create cluster
```

Expected output:

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

kubectl cluster-info --context kind-kind

Thanks for using kind!
```

Check whether the cluster is successfully created:

```
kubectl cluster-info
```

Expected output:

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

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

You're now ready to deploy TiDB Operator.

To destroy the Kubernetes cluster, run the following command:

```
kind delete cluster
```

3.1.2 Create a Kubernetes cluster using minikube

This section describes how to deploy a Kubernetes cluster using minikube.

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

> **Warning:**
>
> The minikube cluster is for test purposes only. **Do not use** it for production.

Before deployment, make sure the following requirements are satisfied:

- minikube: version 1.0.0+
  - minikube requires a compatible hypervisor. For details, refer to minikube's installation instructions.
- kubectl: version >= 1.12

> **Note:**
>
> Although minikube supports `--vm-driver=none` that uses host Docker instead of VM, it is not fully tested with TiDB Operator and might not work. If you want to try TiDB Operator on a system without virtualization support (for example, on a VPS), you might consider using kind instead.

After minikube is installed, execute the following command to start a minikube Kubernetes cluster:

```
minikube start
```

You should see output like this, with some differences depending on your OS and hypervisor:

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

  $ sudo chown root:wheel /Users/user/.minikube/bin/docker-machine-driver-
  ↪ hyperkit
  $ sudo chmod u+s /Users/user/.minikube/bin/docker-machine-driver-
  ↪ hyperkit


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

If you have trouble accessing Docker Hub, you might use your 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:

Because minikube is running on VMs (by default), `127.0.0.1` is the IP address of the VM itself. You might need to modify the proxy to use the real IP address of the host machine in some cases.

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

To interact with the cluster, you can use `kubectl`, which is included as a sub-command in `minikube`. To make the `kubectl` command available, you can either add the following alias definition command to your shell profile or execute the following alias definition command after opening a new shell.

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

Execute this command to check the status of your Kubernetes and make sure `kubectl` can connect to it:

```
kubectl cluster-info
```

Expect this output:

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

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

You are now ready to deploy TiDB Operator.

To destroy the Kubernetes cluster, run the following command:

```
minikube delete
```

3.2 Deploy TiDB Operator

Before proceeding, make sure the following requirements are satisfied:

- A running Kubernetes cluster that `kubectl` can connect to
- Helm 3 is installed

Deploy TiDB Operator takes two steps:

- Install TiDB Operator CRDs
- Install TiDB Operator

3.2.1 Install TiDB Operator CRDs

TiDB Operator includes a number of Custom Resource Definitions (CRDs) that implement different components of the TiDB cluster.

Execute this command to install the CRDs into your cluster:

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

Expected output:

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

3.2.2 Install TiDB Operator

This section describes how to install TiDB Operator using Helm 3.

1. Add the PingCAP repository:

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

   Expected output:

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

2. Create a namespace for TiDB Operator:

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

Expected output:

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

3. Install TiDB Operator

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

If you have trouble accessing Docker Hub, you can try images hosted in Alibaba Cloud:

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

Expected output:

```
NAME: tidb-operator
LAST DEPLOYED: Mon Jun 1 12:31:43 2020
NAMESPACE: tidb-admin
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Make sure tidb-operator components are running:

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

To confirm that the TiDB Operator components are running, execute the following command:

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

Expected output:

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

As soon as all Pods are in the "Running" state, proceed to the next step.

3.3 Deploy a TiDB cluster and its monitoring services

This section describes how to deploy a TiDB cluster and its monitoring services.

3.3.1 Deploy a TiDB cluster

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

If you have trouble accessing Docker Hub, you can try images hosted in Alibaba Cloud:

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

Expected output:

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

> **Note:**
>
> If you need to deploy a TiDB cluster on ARM64 machines, refer to Deploy a TiDB Cluster on ARM64 Machines.

3.3.2 Deploy TiDB monitoring services

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

If you have trouble accessing Docker Hub, you can try images hosted in Alibaba Cloud:

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

Expected output:

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

3.3.3 View the Pod status

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

Expected output:

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

Wait until all Pods for all services are started. As soon as you see Pods of each type (`-pd`, `-tikv`, and `-tidb`) are in the "Running" state, you can press Ctrl+C to get back to the command line and go on to connect to your TiDB cluster.

Expected output:

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

3.4 Connect to TiDB

Because TiDB supports the MySQL protocol and most of its syntax, you can connect to TiDB using the MySQL client.

3.4.1 Install the MySQL client

To connect to TiDB, you need a MySQL-compatible client installed on the host where `kubectl` is installed. This can be the `mysql` executable from an installation of MySQL Server, MariaDB Server, Percona Server, or a standalone client executable from your operating system's package repository.

3.4.2 Forward port 4000

You can connect to TiDB by first forwarding a port from the local host to the TiDB **service** in Kubernetes.

First, get a list of services in the `tidb-cluster` namespace:

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

Expected output:

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

In this case, the TiDB **service** is called **basic-tidb**. Run the following command to forward this port from the local host to the cluster:

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

This command runs in the background and writes its output to a file called `pf4000.out`, so you can continue working in the same shell session.

3.4.3 Connect to the TiDB service

Note:

To connect to TiDB (< v4.0.7) using a MySQL 8.0 client, if the user account has a password, you must explicitly specify `--default-auth=` ↪ `mysql_native_password`. This is because `mysql_native_password` is no longer the default plugin.

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

Expected output:

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

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

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

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

mysql>
```

After connecting to the cluster, you can execute the following commands to verify some of the functionality available in TiDB. (Some of these require TiDB 4.0; if you've deployed an earlier version, upgrade by consulting the Upgrade the TiDB cluster section).

- Create a `hello_world` table:

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

mysql> select * from information_schema.tikv_region_status where
    ↪ db_name=database() and table_name='hello_world'\G
*************************** 1. row ***************************
              REGION_ID: 2
              START_KEY: 7480000000000000FF3500000000000000F8
                END_KEY:
               TABLE_ID: 53
                DB_NAME: test
             TABLE_NAME: hello_world
               IS_INDEX: 0
               INDEX_ID: NULL
             INDEX_NAME: NULL
         EPOCH_CONF_VER: 1
          EPOCH_VERSION: 26
          WRITTEN_BYTES: 0
```

```
           READ_BYTES: 0
       APPROXIMATE_SIZE: 1
       APPROXIMATE_KEYS: 0
REPLICATIONSTATUS_STATE: NULL
REPLICATIONSTATUS_STATEID: NULL
1 row in set (0.011 sec)
```

- Query the TiDB version:

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

- Query the TiKV store status:

```
mysql> select * from information_schema.tikv_store_status\G
*************************** 1. row ***************************
       STORE_ID: 1
       ADDRESS: basic-tikv-0.basic-tikv-peer.tidb-cluster.svc:20160
   STORE_STATE: 0
STORE_STATE_NAME: Up
          LABEL: null
       VERSION: 5.0.6
       CAPACITY: 49.98GiB
       AVAILABLE: 28.47GiB
   LEADER_COUNT: 26
   LEADER_WEIGHT: 1
   LEADER_SCORE: 26
   LEADER_SIZE: 26
   REGION_COUNT: 26
   REGION_WEIGHT: 1
   REGION_SCORE: 4299796.044762109
   REGION_SIZE: 26
       START_TS: 2022-02-17 06:13:46
LAST_HEARTBEAT_TS: 2022-02-17 06:18:16
       UPTIME: 4m30.708704931s
```

```
1 row in set (0.003 sec)
```

- Query the TiDB cluster information:

  (This command requires TiDB 4.0 or later versions. If you've deployed an earlier version, upgrade the TiDB cluster.)

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

3.4.4 Access Grafana dashboard

You can forward the port for Grafana so that you can access Grafana dashboard locally:

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

You can access Grafana dashboard at http://localhost:3000 on the host where you run `kubectl`. Note that if you are not running `kubectl` on the same host (for example, in

a Docker container or on a remote host), you cannot access Grafana dashboard at http://localhost:3000 from your browser.

The default username and password in Grafana are both `admin`.

For more information about monitoring the TiDB cluster in TiDB Operator, refer to Deploy Monitoring and Alerts for a TiDB Cluster.

3.5 Upgrade a TiDB cluster

TiDB Operator also makes it easy to perform a rolling upgrade of the TiDB cluster. This section describes how to upgrade your TiDB cluster to the "nightly" release.

Before that, first you need to get familiar with two `kubectl` sub-commands:

- `kubectl edit` opens a resource specification in an interactive text editor, where an administrator can make changes and save them. If the changes are valid, they'll be propagated to the cluster resources; if they're invalid, they'll be rejected with an error message. Note that not all elements of the specification are validated at this time; it's possible to save changes that may not be applied to the cluster even though they're accepted.
- `kubectl patch` applies a specification change directly to the running cluster resources. There are several different patch strategies, each of which has various capabilities, limitations, and allowed formats.

3.5.1 Modify the TiDB cluster version

In this case, you can use a JSON merge patch to update the version of the TiDB cluster to "nightly":

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

Expected output:

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

3.5.2 Wait for Pods to restart

To follow the progress of the cluster as its components are upgraded, execute the following command. You should see some Pods transiting to "Terminating" and then back to "ContainerCreating" and then to "Running". Note that the value in the "AGE" pod column to see which pods have restarted.

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

Expected output:

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

3.5.3 Forward the TiDB service port

After all Pods have been restarted, you should be able to see that the version number of the cluster has changed.

Note that any port forwarding you set up in a previous step will need to be re-done, because the pod(s) they forwarded to will have been destroyed and re-created. If the `kubectl port-forward` process is still running in your shell, kill it before forwarding the port again.

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

3.5.4 Check the TiDB cluster version

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

Expected output:

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

Note:

`release-4.0-nightly` is not a fixed version. Running the command above at different time might return different results.

For more details about upgrading the TiDB cluster running in TiDB Operator, refer to Upgrade the TiDB cluster.

3.6 Destroy a TiDB cluster

After you've finished testing, you may wish to destroy the TiDB cluster.

Instructions for destroying the Kubernetes cluster depend on how the cluster is created. Refer to Create a Kubernetes test cluster for more details.

The following steps show how to destroy the TiDB cluster, but do not affect the Kubernetes cluster itself.

3.6.1 Delete the TiDB cluster

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

The `tc` in this command is a short name for tidbclusters.

3.6.2 Delete TiDB monitoring services

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

3.6.3 Delete PV data

If your deployment has persistent data storage, deleting the TiDB cluster does not remove the cluster's data. If you do not need the data anymore, run the following commands to clean the data and the dynamically created persistent disks:

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

3.6.4 Delete namespaces

To make sure there are no lingering resources, you can delete the namespace used for your TiDB cluster.

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

3.6.5 Stop `kubectl` port forwarding

If you still have running `kubectl` processes that are forwarding ports, end them:

```
pgrep -lfa kubectl
```

For more information about destroying a TiDB cluster running in TiDB Operator, consult Destroy a TiDB Cluster.

3.7 What's next

If you are ready to deploy a TiDB cluster in Kubernetes for the production environment, refer to the following documents:

- Deploy TiDB in General Kubernetes
- Deploy TiDB on AWS EKS
- Deploy TiDB on GCP GKE
- Deploy TiDB on Alibaba Cloud ACK

4 Deploy

4.1 Deploy TiDB Cluster

4.1.1 Deploy TiDB on AWS EKS

This document describes how to deploy a TiDB cluster on AWS Elastic Kubernetes Service (EKS).

To deploy TiDB Operator and the TiDB cluster in a self-managed Kubernetes environment, refer to Deploy TiDB Operator and Deploy TiDB in General Kubernetes.

4.1.1.1 Prerequisites

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

- Install Helm 3: used for deploying TiDB Operator.
- Complete all operations in Getting started with eksctl.

  This guide includes the following contents:

  - Install and configure `awscli`.
  - Install and configure `eksctl` used for creating Kubernetes clusters.
  - Install `kubectl`.

To verify whether AWS CLI is configured correctly, run the `aws configure list` command. If the output shows the values for `access_key` and `secret_key`, AWS CLI is configured correctly. Otherwise, you need to re-configure AWS CLI.

> **Note:**
>
> The operations described in this document require at least the minimum privileges needed by `eksctl` and the service privileges needed to create a Linux bastion host.

4.1.1.2 Recommended instance types and storage

- Instance types: to gain better performance, the following is recommended:
  - PD nodes: `c5.xlarge`
  - TiDB nodes: `c5.4xlarge`
  - TiKV or TiFlash nodes: `m5.4xlarge`
- Storage: Because AWS supports the EBS `gp3` volume type, it is recommended to use EBS `gp3`. For `gp3` provisioning, the following is recommended:
  - TiKV: 400 MiB/s, 4000 IOPS
  - TiFlash: 625 MiB/s, 6000 IOPS
- AMI type: Amazon Linux 2

4.1.1.3 Create an EKS cluster and a node pool

According to AWS Official Blog recommendation and EKS Best Practice Document, since most of the TiDB cluster components use EBS volumes as storage, it is recommended to create a node pool in each availability zone (at least 3 in total) for each component when creating an EKS.

Save the following configuration as the `cluster.yaml` file. Replace `${clusterName}` with your desired cluster name. The cluster and node group names should match the regular expression `[a-zA-Z][-a-zA-Z0-9]*`, so avoid names that contain `_`.

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

nodeGroups:
  - name: admin
```

```
  desiredCapacity: 1
  privateNetworking: true
  labels:
    dedicated: admin

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

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

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

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

By default, only two TiDB nodes are required, so you can set the `desiredCapacity` of

the `tidb-1d` node group to `0`. You can scale out this node group any time if necessary.

Execute the following command to create the cluster:

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

After executing the command above, you need to wait until the EKS cluster is successfully created and the node group is created and added in the EKS cluster. This process might take 5 to 20 minutes. For more cluster configuration, refer to `eksctl` documentation.

> **Warning:**
>
> If the Regional Auto Scaling Group (ASG) is used:
>
> - Enable the instance scale-in protection for all the EC2s that have been started. The instance scale-in protection for the ASG is not required.
> - Set termination policy to `NewestInstance` for the ASG.

4.1.1.4 Configure StorageClass

This section describes how to configure the storage class for different storage types. These storage types are:

- The default `gp2` storage type after creating the EKS cluster.
- The `gp3` storage type (recommended) or other EBS storage types.
- The local storage used for testing bare-metal performance.

4.1.1.4.1 Configure gp2

After you create an EKS cluster, the default StorageClass is `gp2`. To improve I/O write performance, it is recommended to configure `nodelalloc` and `noatime` in the `mountOptions` field of the `StorageClass` resource.

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

For more information on the mount options, see TiDB Environment and System Configuration Check.

4.1.1.4.2 Configure gp3 (recommended) or other EBS storage types

If you do not want to use the default `gp2` storage type, you can create StorageClass for other storage types. For example, you can use the `gp3` (recommended) or `io1` storage type.

The following example shows how to create and configure a StorageClass for the `gp3` storage type:

1. Deploy the AWS EBS Container Storage Interface (CSI) driver on the EKS cluster. If you are using a storage type other than `gp3`, skip this step.

2. Create a `StorageClass` resource. In the resource definition, specify your desired storage type in the `parameters.type` field.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gp3
provisioner: ebs.csi.aws.com
allowVolumeExpansion: true
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  fsType: ext4
  iops: "4000"
  throughput: "400"
mountOptions:
- nodelalloc,noatime
```

3. In the TidbCluster YAML file, configure `gp3` in the `storageClassName` field. For example:

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

4. To improve I/O write performance, it is recommended to configure `nodelalloc` and `noatime` in the `mountOptions` field of the `StorageClass` resource.

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

For more information on the mount options, see TiDB Environment and System Configuration Check.

For more information on the EBS storage types and configuration, refer to Amazon EBS volume types and Storage Classes.

4.1.1.4.3 Configure local storage

Local storage is used for testing bare-metal performance. For higher IOPS and lower latency, you can choose NVMe SSD volumes offered by some AWS instances for the TiKV node pool. However, for the production environment, use AWS EBS as your storage type.

Note:

- You cannot dynamically change StorageClass for a running TiDB cluster. For testing purposes, create a new TiDB cluster with the desired StorageClass.
- EKS upgrade or other reasons might cause node reconstruction. In such cases, data in the local storage might be lost. To avoid data loss, you need to back up TiKV data before node reconstruction.
- To avoid data loss from node reconstruction, you can refer to AWS documentation and disable the `ReplaceUnhealthy` feature of the TiKV node group.

For instance types that provide NVMe SSD volumes, check out Amazon EC2 Instance Types.

The following `c5d.4xlarge` example shows how to configure StorageClass for the local storage:

1. Create a node group with local storage for TiKV.

    1. In the `eksctl` configuration file, modify the instance type of the TiKV node group to `c5d.4xlarge`:

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

```
  taints:
    dedicated: tikv:NoSchedule
  ...
```

2. Create a node group with local storage:

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

If the TiKV node group already exists, to avoid name conflict, you can take either of the following actions:

- Delete the old group and create a new one.
- Change the group name.

2. Deploy local volume provisioner.

1. To conveniently discover and manage local storage volumes, install local-volume-provisioner.
2. Mount the local storage to the `/mnt/ssd` directory.
3. According to the mounting configuration, modify the local-volume-provisioner.yaml file.
4. Deploy and create a `local-storage` storage class using the modified `local-volume-provisioner.yaml` file.

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

3. Use the local storage.

After you complete the previous step, local-volume-provisioner can discover all the local NVMe SSD volumes in the cluster.

After local-volume-provisioner discovers the local volumes, when you Deploy a TiDB cluster and the monitoring component, you need to add the `tikv.storageClassName` field to `tidb-cluster.yaml` and set the field value to `local-storage`.

4.1.1.5 Deploy TiDB Operator

To deploy TiDB Operator in the EKS cluster, refer to the *Deploy TiDB Operator* section in Getting Started.

4.1.1.6 Deploy a TiDB cluster and the monitoring component

This section describes how to deploy a TiDB cluster and its monitoring component in AWS EKS.

4.1.1.6.1 Create namespace

To create a namespace to deploy the TiDB cluster, run the following command:

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

> **Note:**
>
> A namespace is a virtual cluster backed by the same physical cluster. This document takes `tidb-cluster` as an example. If you want to use another namespace, modify the corresponding arguments of `-n` or `--namespace`.

4.1.1.6.2 Deploy

First, download the sample `TidbCluster` and `TidbMonitor` configuration files:

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

Refer to configure the TiDB cluster to further customize and configure the CR before applying.

> **Note:**
>
> By default, the configuration in `tidb-cluster.yaml` sets up the LoadBalancer for TiDB with the "internal" scheme. This means that the LoadBalancer is only accessible within the VPC, not externally. To access TiDB over the MySQL protocol, you need to use a bastion host or use `kubectl port-forward`. If you want to expose TiDB over the internet and if you are aware of the risks of doing this, you can change the scheme for the LoadBalancer from "internet-facing" in the `tidb-cluster.yaml` file.

To deploy the `TidbCluster` and `TidbMonitor` CR in the EKS cluster, run the following command:

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

After the YAML file above is applied to the Kubernetes cluster, TiDB Operator creates the desired TiDB cluster and its monitoring component according to the YAML file.

Note:

If you need to deploy a TiDB cluster on ARM64 machines, refer to Deploy a TiDB Cluster on ARM64 Machines.

4.1.1.6.3 View the cluster status

To view the status of the starting TiDB cluster, run the following command:

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

When all the Pods are in the `Running` or `Ready` state, the TiDB cluster is successfully started. For example:

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

4.1.1.7 Access the database

After you have deployed a TiDB cluster, you can access the TiDB database to test or develop your application.

4.1.1.7.1 Prepare a bastion host

The LoadBalancer created for your TiDB cluster is an intranet LoadBalancer. You can create a bastion host in the cluster VPC to access the database. To create a bastion host on AWS console, refer to AWS documentation.

Select the cluster's VPC and Subnet, and verify whether the cluster name is correct in the dropdown box. You can view the cluster's VPC and Subnet by running the following command:

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

Allow the bastion host to access the Internet. Select the correct key pair so that you can log in to the host via SSH.

> **Note:**
>
> In addition to the bastion host, you can also connect an existing host to the cluster VPC by VPC Peering. If the EKS cluster is created in an existing VPC, you can use the host in the VPC.

4.1.1.7.2 Install the MySQL client and connect

After the bastion host is created, you can connect to the bastion host via SSH and access the TiDB cluster via the MySQL client.

1. Log in to the bastion host via SSH:

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

2. Install the MySQL client on the bastion host:

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

3. Connect the client to the TiDB cluster:

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

`${tidb-nlb-dnsname}` is the LoadBalancer domain name of the TiDB service. You can view the domain name in the `EXTERNAL-IP` field by executing `kubectl get svc ↪ basic-tidb -n tidb-cluster`.

For example:

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

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

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

```
MySQL [(none)]> show status;
+--------------------+--------------------------------------+
| Variable_name    | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher       |                                      |
| Ssl_cipher_list  |                                      |
| Ssl_verify_mode  | 0                                    |
| Ssl_version      |                                      |
| ddl_schema_version | 22                                   |
| server_id        | ed4ba88b-436a-424d-9087-977e897cf5ec |
+--------------------+--------------------------------------+
6 rows in set (0.00 sec)
```

Note:

- The default authentication plugin of MySQL 8.0 is updated from `mysql_native_password` to `caching_sha2_password`. Therefore, if you use MySQL client from MySQL 8.0 to access the TiDB service (cluster version < v4.0.7), and if the user account has a password, you need to explicitly specify the `--default-auth=mysql_native_password` parameter.
- By default, TiDB (starting from v4.0.2) periodically shares usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see Telemetry.

4.1.1.8 Access the Grafana monitoring dashboard

Obtain the LoadBalancer domain name of Grafana:

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

For example:

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

In the output above, the `EXTERNAL-IP` column is the LoadBalancer domain name.

You can access the `${grafana-lb}:3000` address using your web browser to view monitoring metrics. Replace `${grafana-lb}` with the LoadBalancer domain name.

> **Note:**
>
> The default Grafana username and password are both `admin`.

4.1.1.9 Access the TiDB Dashboard

See Access TiDB Dashboard for instructions about how to securely allow access to the TiDB Dashboard.

4.1.1.10 Upgrade

To upgrade the TiDB cluster, edit the `spec.version` by executing `kubectl edit tc basic -n tidb-cluster`.

The upgrade process does not finish immediately. You can watch the upgrade progress by executing `kubectl get pods -n tidb-cluster --watch`.

4.1.1.11 Scale out

Before scaling out the cluster, you need to scale out the corresponding node group so that the new instances have enough resources for operation.

This section describes how to scale out the EKS node group and TiDB components.

4.1.1.11.1 Scale out EKS node group

When scaling out TiKV, the node groups must be scaled out evenly among the different availability zones. The following example shows how to scale out the `tikv-1a`, `tikv-1c`, and `tikv-1d` groups of the `${clusterName}` cluster to 2 nodes:

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

For more information on managing node groups, refer to `eksctl` documentation.

4.1.1.11.2 Scale out TiDB components

After scaling out the EKS node group, execute `kubectl edit tc basic -n tidb-cluster`, and modify each component's `replicas` to the desired number of replicas. The scaling-out process is then completed.

4.1.1.12 Deploy TiFlash/TiCDC

TiFlash is the columnar storage extension of TiKV.

TiCDC is a tool for replicating the incremental data of TiDB by pulling TiKV change logs.

The two components are *not required* in the deployment. This section shows a quick start example.

4.1.1.12.1 Add node groups

In the configuration file of eksctl (`cluster.yaml`), add the following two items to add a node group for TiFlash/TiCDC respectively. `desiredCapacity` is the number of nodes you desire.

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

- name: ticdc-1a
```

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

Depending on the EKS cluster status, use different commands:

- If the cluster is not created, execute `eksctl create cluster -f cluster.yaml` to create the cluster and node groups.
- If the cluster is already created, execute `eksctl create nodegroup -f cluster.yaml` to create the node groups. The existing node groups are ignored and will not be created again.

4.1.1.12.2 Configure and deploy

- To deploy TiFlash, configure `spec.tiflash` in `tidb-cluster.yaml`:

```
spec:
  ...
  tiflash:
    baseImage: pingcap/tiflash
    replicas: 1
    storageClaims:
    - resources:
        requests:
```

```
        storage: 100Gi
  tolerations:
  - effect: NoSchedule
    key: dedicated
    operator: Equal
    value: tiflash
```

For other parameters, refer to Configure a TiDB Cluster.

> **Warning:**
> TiDB Operator automatically mount PVs **in the order of the configuration** in the `storageClaims` list. Therefore, if you need to add disks for TiFlash, make sure that you add the disks **only to the end of the original configuration** in the list. In addition, you must **not** alter the order of the original configuration.

- To deploy TiCDC, configure `spec.ticdc` in `tidb-cluster.yaml`:

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

Modify `replicas` according to your needs.

Finally, execute `kubectl -n tidb-cluster apply -f tidb-cluster.yaml` to update the TiDB cluster configuration.

For detailed CR configuration, refer to API references and Configure a TiDB Cluster.

4.1.2 Deploy TiDB on GCP GKE

This document describes how to deploy a GCP Google Kubernetes Engine (GKE) cluster and deploy a TiDB cluster on GCP GKE.

To deploy TiDB Operator and the TiDB cluster in a self-managed Kubernetes environment, refer to Deploy TiDB Operator and Deploy TiDB in General Kubernetes.

4.1.2.1 Prerequisites

Before deploying a TiDB cluster on GCP GKE, make sure the following requirements are satisfied:

- Install Helm 3: used for deploying TiDB Operator.
- Install gcloud: a command-line tool used for creating and managing GCP services.
- Complete the operations in the **Before you begin** section of GKE Quickstart.

  This guide includes the following contents:

  - Enable Kubernetes APIs
  - Configure enough quota

4.1.2.2 Recommended instance types and storage

- Instance types: to gain better performance, the following is recommended:
  - PD nodes: `n2-standard-4`
  - TiDB nodes: `n2-standard-16`
  - TiKV or TiFlash nodes: `n2-standard-16`
- Storage: For TiKV or TiFlash, it is recommended to use pd-ssd disk type.

4.1.2.3 Configure the GCP service

Configure your GCP project and default region:

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

4.1.2.4 Create a GKE cluster and node pool

1. Create a GKE cluster and a default node pool:

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

   - The command above creates a regional cluster.
   - The `--num-nodes=1` option indicates that one node is created in each zone. So if there are three zones in the region, there are three nodes in total, which ensures high availability.
   - It is recommended to use regional clusters in production environments. For other types of clusters, refer to Types of GKE clusters.

- The command above creates a cluster in the default network. If you want to specify a network, use the `--network/subnet` option. For more information, refer to Creating a regional cluster.

2. Create separate node pools for PD, TiKV, and TiDB:

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

The process might take a few minutes.

4.1.2.5 Configure StorageClass

After the GKE cluster is created, the cluster contains three StorageClasses of different disk types.

- standard: `pd-standard` disk type (default)
- standard-rwo: `pd-balanced` disk type
- premium-rwo: `pd-ssd` disk type (recommended)

To improve I/O write performance, it is recommended to configure `nodelalloc` and `noatime` in the `mountOptions` field of the StorageClass resource. For details, see TiDB Environment and System Configuration Check.

It is recommended to use the default `pd-ssd` storage class `premium-rwo` or to set up a customized storage class:

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

Note:

Configuring `nodelalloc` and `noatime` is not supported for the default disk type `pd-standard`.

4.1.2.5.1 Use local storage

For the production environment, use zonal persistent disks.

If you need to simulate bare-metal performance, some GCP instance types provide additional local store volumes. You can choose such instances for the TiKV node pool to achieve higher IOPS and lower latency.

Note:

You cannot dynamically change StorageClass for a running TiDB cluster. For testing purposes, create a new TiDB cluster with the desired StorageClass.

GKE upgrade might cause node reconstruction. In such cases, data in the local storage might be lost. To avoid data loss, you need to back up TiKV data before node reconstruction. It is thus not recommended to use local disks in the production environment.

1. Create a node pool with local storage for TiKV:

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

   If the TiKV node pool already exists, you can either delete the old pool and then create a new one, or change the pool name to avoid conflict.

2. Deploy the local volume provisioner.

   You need to use the local-volume-provisioner to discover and manage the local storage. Executing the following command deploys and creates a `local-storage` storage class:

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

3. Use the local storage.

   After the steps above, the local volume provisioner can discover all the local NVMe SSD disks in the cluster.

   Modify `tikv.storageClassName` in the `tidb-cluster.yaml` file to `local-storage`.

4.1.2.6 Deploy TiDB Operator

To deploy TiDB Operator on GKE, refer to deploy TiDB Operator.

4.1.2.7 Deploy a TiDB cluster and the monitoring component

This section describes how to deploy a TiDB cluster and its monitoring component on GCP GKE.

4.1.2.7.1 Create namespace

To create a namespace to deploy the TiDB cluster, run the following command:

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

> **Note:**
>
> A namespace is a virtual cluster backed by the same physical cluster. This document takes `tidb-cluster` as an example. If you want to use other namespace, modify the corresponding arguments of `-n` or `--namespace`.

4.1.2.7.2 Deploy

First, download the sample `TidbCluster` and `TidbMonitor` configuration files:

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

Refer to configure the TiDB cluster to further customize and configure the CR before applying.

To deploy the `TidbCluster` and `TidbMonitor` CR in the GKE cluster, run the following command:

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

After the yaml file above is applied to the Kubernetes cluster, TiDB Operator creates the desired TiDB cluster and its monitoring component according to the yaml file.

> **Note:**
>
> If you need to deploy a TiDB cluster on ARM64 machines, refer to Deploy a TiDB Cluster on ARM64 Machines.

4.1.2.7.3 View the cluster status

To view the status of the starting TiDB cluster, run the following command:

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

When all the Pods are in the `Running` or `Ready` state, the TiDB cluster is successfully started. For example:

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

4.1.2.8 Access the TiDB database

After you deploy a TiDB cluster, you can access the TiDB database via MySQL client.

4.1.2.8.1 Prepare a bastion host

The LoadBalancer created for your TiDB cluster is an intranet LoadBalancer. You can create a bastion host in the cluster VPC to access the database.

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

Note:

`${your-region}-a` is the `a` zone in the region of the cluster, such as `us-central1-a`. You can also create the bastion host in other zones in the same region.

4.1.2.8.2 Install the MySQL client and connect

After the bastion host is created, you can connect to the bastion host via SSH and access the TiDB cluster via the MySQL client.

1. Connect to the bastion host via SSH:

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

2. Install the MySQL client:

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

3. Connect the client to the TiDB cluster:

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

`${tidb-nlb-dnsname}` is the LoadBalancer IP of the TiDB service. You can view the IP in the `EXTERNAL-IP` field of the `kubectl get svc basic-tidb -n tidb-cluster` execution result.

For example:

```
$ mysql -h 10.128.15.243 -P 4000 -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 7823
Server version: 5.7.25-TiDB-v4.0.4 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 5.7 compatible

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

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

MySQL [(none)]> show status;
+--------------------+--------------------------------------+
| Variable_name    | Value                                |
+--------------------+--------------------------------------+
```

```
| Ssl_cipher       |                                      |
| Ssl_cipher_list  |                                      |
| Ssl_verify_mode  | 0                                    |
| Ssl_version      |                                      |
| ddl_schema_version | 22                                 |
| server_id        | 717420dc-0eeb-4d4a-951d-0d393aff295a |
+--------------------+--------------------------------------+
6 rows in set (0.01 sec)
```

Note:

- The default authentication plugin of MySQL 8.0 is updated from `mysql_native_password` to `caching_sha2_password`. Therefore, if you use MySQL client from MySQL 8.0 to access the TiDB service (TiDB version < v4.0.7), and if the user account has a password, you need to explicitly specify the `--default-auth=mysql_native_password` parameter.
- By default, TiDB (starting from v4.0.2) periodically shares usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see Telemetry.

4.1.2.8.3 Access the Grafana monitoring dashboard

Obtain the LoadBalancer IP of Grafana:

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

For example:

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

In the output above, the `EXTERNAL-IP` column is the LoadBalancer IP.

You can access the `${grafana-lb}:3000` address using your web browser to view monitoring metrics. Replace `${grafana-lb}` with the LoadBalancer IP.

Note:

The default Grafana username and password are both `admin`.

4.1.2.9 Upgrade

To upgrade the TiDB cluster, edit the `spec.version` by executing `kubectl edit tc basic -n tidb-cluster`.

The upgrade process does not finish immediately. You can watch the upgrade progress by executing `kubectl get pods -n tidb-cluster --watch`.

4.1.2.10 Scale out

Before scaling out the cluster, you need to scale out the corresponding node pool so that the new instances have enough resources for operation.

This section describes how to scale out the EKS node group and TiDB components.

4.1.2.10.1 Scale out GKE node group

The following example shows how to scale out the `tikv` node pool of the `tidb` cluster to 6 nodes:

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

Note:

In the regional cluster, the nodes are created in 3 zones. Therefore, after scaling out, the number of nodes is `2 * 3 = 6`.

4.1.2.10.2 Scale out TiDB components

After that, execute `kubectl edit tc basic -n tidb-cluster` and modify each component's `replicas` to the desired number of replicas. The scaling-out process is then completed.

For more information on managing node pools, refer to GKE Node pools.

4.1.2.11 Deploy TiFlash and TiCDC

TiFlash is the columnar storage extension of TiKV.

TiCDC is a tool for replicating the incremental data of TiDB by pulling TiKV change logs.

The two components are *not required* in the deployment. This section shows a quick start example.

4.1.2.11.1 Create new node pools

- Create a node pool for TiFlash:

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

- Create a node pool for TiCDC:

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

4.1.2.11.2 Configure and deploy

- To deploy TiFlash, configure `spec.tiflash` in `tidb-cluster.yaml`. For example:

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

To configure other parameters, refer to Configure a TiDB Cluster.

> **Warning:**
> TiDB Operator automatically mounts PVs **in the order of the configuration** in the `storageClaims` list. Therefore, if you need to add disks for TiFlash, make sure that you add the disks **only to the end of the original configuration** in the list. In addition, you must **not** alter the order of the original configuration.

- To deploy TiCDC, configure `spec.ticdc` in `tidb-cluster.yaml`. For example:

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

  Modify `replicas` according to your needs.

Finally, execute `kubectl -n tidb-cluster apply -f tidb-cluster.yaml` to update the TiDB cluster configuration.

For detailed CR configuration, refer to API references and Configure a TiDB Cluster.

4.1.3 Deploy TiDB on Azure AKS

This document describes how to deploy a TiDB cluster on Azure Kubernetes Service (AKS).

To deploy TiDB Operator and the TiDB cluster in a self-managed Kubernetes environment, refer to Deploy TiDB Operator and Deploy TiDB in General Kubernetes.

4.1.3.1 Prerequisites

Before deploying a TiDB cluster on Azure AKS, perform the following operations:

- Install Helm 3 for deploying TiDB Operator.

- Deploy a Kubernetes (AKS) cluster and install and configure `az cli`.

> **Note:**
> To verify whether AZ CLI is configured correctly, run the `az login` command. If login with account credentials succeeds, AZ CLI is configured correctly. Otherwise, you need to re-configure AZ CLI.

- Refer to use Ultra disks to create a new cluster that can use Ultra disks or enable Ultra disks in an exist cluster.
- Acquire AKS service permissions.
- If the Kubernetes version of the cluster is earlier than 1.21, install aks-preview CLI extension for using Ultra Disks and register EnableAzureDiskFileCSIDriver in your subscription.
  - Install the aks-preview CLI extension:

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

  - Register `EnableAzureDiskFileCSIDriver`:

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

4.1.3.2 Create an AKS cluster and a node pool

Most of the TiDB cluster components use Azure disk as storage. According to AKS Best Practices, when creating an AKS cluster, it is recommended to ensure that each node pool uses one availability zone (at least 3 in total).

4.1.3.2.1 Create an AKS cluster with CSI enabled

To create an AKS cluster with CSI enabled, run the following command:

> **Note:**
> If the Kubernetes version of the cluster is earlier than 1.21, you need to append an `--aks-custom-headers` flag to enable the **EnableAzureDiskFileCSIDriver** feature by running the following command:

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

4.1.3.2.2 Create component node pools

After creating an AKS cluster, run the following commands to create component node pools. Each node pool may take two to five minutes to create. It is recommended to enable Ultra disks in the TiKV node pool. For more details about cluster configuration, refer to `az aks` documentation and `az aks nodepool` documentation.

1. To create a TiDB Operator and monitor pool:

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

2. Create a PD node pool with `nodeType` being `Standard_F4s_v2` or higher:

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

3. Create a TiDB node pool with `nodeType` being `Standard_F8s_v2` or higher. You can set `--node-count` to 2 because only two TiDB nodes are required by default. You can also scale out this node pool by modifying this parameter at any time if necessary.

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

4. Create a TiKV node pool with `nodeType` being `Standard_E8s_v4` or higher:

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

4.1.3.2.3 Deploy component node pools in availability zones

The Azure AKS cluster deploys nodes across multiple zones using "best effort zone balance". If you want to apply "strict zone balance" (not supported in AKS now), you can deploy one node pool in one zone. For example:

1. Create TiKV node pool 1 in zone 1:

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

2. Create TiKV node pool 2 in zone 2:

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

3. Create TiKV node pool 3 in zone 3:

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

Warning:

About node pool scale-in:

- You can manually scale in or out an AKS cluster to run a different number of nodes. When you scale in, nodes are carefully cordoned and drained to minimize disruption to running applications. Refer to Scale the node count in an Azure Kubernetes Service (AKS) cluster.

4.1.3.3 Configure StorageClass

To improve disk IO performance, it is recommended to add `mountOptions` in `StorageClass` to configure `nodelalloc` and `noatime`. Refer to Mount the data disk ext4 filesystem with options on the target machines that deploy TiKV.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
### ...
```

```
mountOptions:
- nodelalloc,noatime
```

4.1.3.4 Deploy TiDB Operator

Deploy TiDB Operator in the AKS cluster by referring to *Deploy TiDB Operator* section.

4.1.3.5 Deploy a TiDB cluster and the monitoring component

This section describes how to deploy a TiDB cluster and its monitoring component in Azure AKS.

4.1.3.5.1 Create namespace

To create a namespace to deploy the TiDB cluster, run the following command:

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

> **Note:**
>
> A namespace is a virtual cluster backed by the same physical cluster. This document takes `tidb-cluster` as an example. If you want to use other namespaces, modify the corresponding arguments of `-n` or `--namespace`.

4.1.3.5.2 Deploy

First, download the sample `TidbCluster` and `TidbMonitor` configuration files:

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

Refer to configure the TiDB cluster to further customize and configure the CR before applying.

> **Note:**
>
> By default, TiDB LoadBalancer in `tidb-cluster.yaml` is set to "internal", indicating that the LoadBalancer is only accessible within the cluster virtual network, not externally. To access TiDB over the MySQL protocol, you need

to use a bastion to access the internal host of the cluster or use `kubectl port-forward`. If you understand the risks of exposing the LoadBalancer publicly, you can delete the following annotation in the `tidb-cluster.yaml` file:

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

After deleting the annotation, the recreated LoadBalancer and its associated TiDB services will be externally accessible.

To deploy the `TidbCluster` and `TidbMonitor` CR in the AKS cluster, run the following command:

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

After the yaml file above is applied to the Kubernetes cluster, TiDB Operator creates the desired TiDB cluster and its monitoring component according to the yaml file.

4.1.3.5.3 View the cluster status

To view the status of the TiDB cluster, run the following command:

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

When all the pods are in the `Running` or `Ready` state, the TiDB cluster is successfully started. For example:

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

4.1.3.6 Access the database

After deploying a TiDB cluster, you can access the TiDB database to test or develop applications.

4.1.3.6.1 Access method

- Access via Bastion

The LoadBalancer created for your TiDB cluster resides in an intranet. You can create a Bastion in the cluster virtual network to connect to an internal host and then access the database.

> **Note:**
>
> In addition to the bastion host, you can also connect an existing host to the cluster virtual network by Peering. If the AKS cluster is created in an existing virtual network, you can use hosts in this virtual network to access the database.

- Access via SSH

You can create the SSH connection to a Linux node to access the database.

- Access via node-shell

You can simply use tools like node-shell to connect to nodes in the cluster, then access the database.

4.1.3.6.2 Access via the MySQL client

After access to the internal host via SSH, you can access the TiDB cluster through the MySQL client.

1. Install the MySQL client on the host:

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

2. Connect the client to the TiDB cluster:

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

`${tidb-lb-ip}` is the LoadBalancer IP address of the TiDB service. To obtain it, run the `kubectl get svc basic-tidb -n tidb-cluster` command. The `EXTERNAL-IP` field returned is the IP address.

For example:

```
$ mysql --comments -h 20.240.0.7 -P 4000 -u root
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 1189
Server version: 5.7.25-TiDB-v4.0.2 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 5.7 compatible

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

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

MySQL [(none)]> show status;
+--------------------+--------------------------------------+
| Variable_name      | Value                                |
+--------------------+--------------------------------------+
| Ssl_cipher         |                                      |
| Ssl_cipher_list    |                                      |
| Ssl_verify_mode    | 0                                    |
| Ssl_version        |                                      |
| ddl_schema_version | 22                                   |
| server_id          | ed4ba88b-436a-424d-9087-977e897cf5ec |
+--------------------+--------------------------------------+
6 rows in set (0.00 sec)
```

Note:

- The default authentication plugin of MySQL 8.0 is updated from `mysql_native_password` to `caching_sha2_password`. Therefore, if you access the TiDB service (earlier than v4.0.7) by using MySQL 8.0 client via password authentication, you need to specify the `--default-auth=mysql_native_password` parameter.
- By default, TiDB (starting from v4.0.2) periodically shares usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see Telemetry.

4.1.3.7 Access the Grafana monitoring dashboard

Obtain the LoadBalancer IP address of Grafana:

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

For example:

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

In the output above, the `EXTERNAL-IP` column is the LoadBalancer IP address.

You can access the `${grafana-lb}:3000` address using your web browser to view monitoring metrics. Replace `${grafana-lb}` with the LoadBalancer IP address.

> **Note:**
>
> The default Grafana username and password are both `admin`.

4.1.3.8 Access TiDB Dashboard

See Access TiDB Dashboard for instructions about how to securely allow access to TiDB Dashboard.

4.1.3.9 Upgrade

To upgrade the TiDB cluster, edit `spec.version` by running the `kubectl edit tc basic -n tidb-cluster` command.

The upgrade process does not finish immediately. You can view the upgrade progress by running the `kubectl get pods -n tidb-cluster --watch` command.

4.1.3.10 Scale out

Before scaling out the cluster, you need to scale out the corresponding node pool so that the new instances have enough resources for operation.

This section describes how to scale out the AKS node pool and TiDB components.

4.1.3.10.1 Scale out AKS node pool

When scaling out TiKV, the node pools must be scaled out evenly among availability zones. The following example shows how to scale out the TiKV node pool of the `${clusterName}` cluster to 6 nodes:

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

For more information on node pool management, refer to `az aks nodepool`.

4.1.3.10.2 Scale out TiDB components

After scaling out the AKS node pool, run the `kubectl edit tc basic -n tidb -cluster` command with `replicas` of each component set to desired value. The scaling-out process is then completed.

4.1.3.11 Deploy TiFlash/TiCDC

TiFlash is the columnar storage extension of TiKV.

TiCDC is a tool for replicating the incremental data of TiDB by pulling TiKV change logs.

The two components are *not required* in the deployment. This section shows a quick start example.

4.1.3.11.1 Add node pools

Add a node pool for TiFlash/TiCDC respectively. You can set `--node-count` as required.

1. Create a TiFlash node pool with `nodeType` being `Standard_E8s_v4` or higher:

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

2. Create a TiCDC node pool with `nodeType` being `Standard_E16s_v4` or higher:

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

4.1.3.11.2 Configure and deploy

- To deploy TiFlash, configure `spec.tiflash` in `tidb-cluster.yaml`. The following is an example:

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

For other parameters, refer to Configure a TiDB Cluster.

> **Warning:**
> TiDB Operator automatically mounts PVs **in the order of the configuration** in the `storageClaims` list. Therefore, if you need to add disks for TiFlash, make sure that you add the disks **only to the end of the original configuration** in the list. In addition, you must **not** alter the order of the original configuration.

- To deploy TiCDC, configure `spec.ticdc` in `tidb-cluster.yaml`. The following is an example:

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

Modify `replicas` as required.

Finally, run the `kubectl -n tidb-cluster apply -f tidb-cluster.yaml` command to update the TiDB cluster configuration.

For detailed CR configuration, refer to API references and Configure a TiDB Cluster.

4.1.3.12 Use other Disk volume types

Azure disks support multiple volume types. Among them, `UltraSSD` delivers low latency and high throughput and can be enabled by performing the following steps:

1. Enable Ultra disks on an existing cluster and create a storage class for `UltraSSD`:

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

   You can add more Driver Parameters as required.

2. In `tidb-cluster.yaml`, specify the `ultra` storage class to apply for the `UltraSSD` volume type through the `storageClassName` field.

   The following is a TiKV configuration example you can refer to:

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

You can use any supported Azure disk type. It is recommended to use `Premium_LRS` or `UltraSSD_LRS`.

For more information about the storage class configuration and Azure disk types, refer to Storage Class documentation and Azure Disk Types.

4.1.3.13 Use local storage

Use Azure LRS disks for storage in production environment. To simulate bare-metal performance, use additional NVMe SSD local store volumes provided by some Azure instances. You can choose such instances for the TiKV node pool to achieve higher IOPS and lower latency.

> **Note:**
>
> - You cannot dynamically change the storage class of a running TiDB cluster. In this case, create a new cluster for testing.
> - Local NVMe Disks are ephemeral. Data will be lost on these disks if you stop/deallocate your node. When the node is reconstructed, you need to migrate data in TiKV. If you do not want to migrate data, it is recommended not to use the local disk in a production environment.

For instance types that provide local disks, refer to Lsv2-series. The following takes `Standard_L8s_v2` as an example:

1. Create a node pool with local storage for TiKV.

   Modify the instance type of the TiKV node pool in the `az aks nodepool add` command to `Standard_L8s_v2`:

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

   If the TiKV node pool already exists, you can either delete the old group and then create a new one, or change the group name to avoid conflict.

2. Deploy the local volume provisioner.

   You need to use the local-volume-provisioner to discover and manage the local storage. Run the following command to deploy and create a `local-storage` storage class:

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

3. Use local storage.

   After the steps above, the local volume provisioner can discover all the local NVMe SSD disks in the cluster.

   Add the `tikv.storageClassName` field to the `tidb-cluster.yaml` file and set the value of the field to `local-storage`.

   For more information, refer to Deploy TiDB cluster and its monitoring components

4.1.4 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.

To deploy TiDB Operator and the TiDB cluster in a self-managed Kubernetes environment, refer to Deploy TiDB Operator and Deploy TiDB in General Kubernetes.

4.1.4.1 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 3
- 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.

4.1.4.1.1 Required privileges

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

- AliyunECSFullAccess
- AliyunESSFullAccess
- AliyunVPCFullAccess
- AliyunSLBFullAccess

- AliyunCSFullAccess
- AliyunEIPFullAccess
- AliyunECIFullAccess
- AliyunVPNGatewayFullAccess
- AliyunNATGatewayFullAccess

4.1.4.2 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). The default auto-scaling group of managed Kubernetes must have at least two instances to host the whole 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.

4.1.4.3 Deploy

4.1.4.3.1 Deploy ACK, TiDB Operator and the node pool for TiDB cluster

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

You can create or modify `terraform.tfvars` to set the values of the variables, and configure the cluster to fit your needs. You can view the configurable variables and their descriptions in `variables.tf`. The following is an example of how to configure the ACK cluster name, the TiDB cluster name, the TiDB Operator version, and the number of PD, TiKV, and TiDB nodes.

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

- To deploy TiFlash in the cluster, set `create_tiflash_node_pool = true` ↪ in `terraform.tfvars`. You can also configure the node count and instance type of the TiFlash node pool by modifying `tiflash_count` and `tiflash_instance_type`. By default, the value of `tiflash_count` is `2`, and the value of `tiflash_instance_type` is `ecs.i2.2xlarge`.
- To deploy TiCDC in the cluster, set `create_cdc_node_pool = true` in `terraform.tfvars`. You can also configure the node count and instance type of the TiCDC node pool by modifying `cdc_count` and `cdc_instance_type`. By default, the value of `cdc_count` is `3`, and the value of `cdc_instance_type` is `ecs.c5.2xlarge`.

> **Note:**
> Check the `operator_version` in the `variables.tf` file for the default TiDB Operator version of the current scripts. If the default version is not your desired one, configure `operator_version` in `terraform.tfvars`.

After the configuration, execute the following commands to initialize and deploy the cluster:

```
terraform init
```

Input "yes" to confirm execution when you run the following `apply` command:

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

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
```

4.1.4.3.2 Deploy the TiDB cluster and monitor

1. Prepare the `TidbCluster` and `TidbMonitor` CR files:

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

To complete the CR file configuration, refer to TiDB Operator API documentation and Configure a TiDB Cluster.

- To deploy TiFlash, configure `spec.tiflash` in `db.yaml` as follows:

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

To configure other parameters, refer to Configure a TiDB Cluster.

Modify `replicas`, `storageClaims[].resources.requests.storage`, and `storageClassName` according to your needs.

> **Warning:**
> Since TiDB Operator will mount PVs automatically in the **order** of the items in the `storageClaims` list, if you need to add more disks to TiFlash, make sure to append the new item only to the **end** of the original items, and **DO NOT** modify the order of the original items.

- To deploy TiCDC, configure `spec.ticdc` in `db.yaml` as follows:

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

```
    key: dedicated
    operator: Equal
    value: TIDB_CLUSTER_NAME-cdc
```

Modify `replicas` according to your needs.

Note:

- Replace all the `TIDB_CLUSTER_NAME` in the `db.yaml` and `db-monitor.yaml` files with `tidb_cluster_name` configured in the deployment of ACK.
- Make sure the number of PD, TiKV, TiFlash, TiCDC, or TiDB nodes is >= the `replicas` value of the corresponding component in `db.yaml`.
- Make sure `spec.initializer.version` in `db-monitor.yaml` is the same as `spec.version` in `db.yaml`. Otherwise, the monitor might not display correctly.

2. Create `Namespace`:

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

Note:

You can give the `namespace` a name that is easy to memorize, such as the same name as `tidb_cluster_name`.

3. Deploy the TiDB cluster:

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

Note:

If you need to deploy a TiDB cluster on ARM64 machines, refer to Deploy a TiDB Cluster on ARM64 Machines.

4.1.4.4 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_lb_ip} -P 4000 -u root
```

`tidb_lb_ip` is the LoadBalancer IP of the TiDB service.

> **Note:**
>
> - The default authentication plugin of MySQL 8.0 is updated from `mysql_native_password` to `caching_sha2_password`. Therefore, if you use MySQL client from MySQL 8.0 to access the TiDB service (TiDB version < v4.0.7), and if the user account has a password, you need to explicitly specify the `--default-auth=mysql_native_password` parameter.
> - By default, TiDB (starting from v4.0.2) periodically shares usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see Telemetry.

4.1.4.5 Monitor

Visit `<monitor-lb>:3000` to view the Grafana dashboards. `monitor-lb` is the LoadBalancer IP of the Monitor service.

The initial login user account and password:

- User: admin
- Password: admin

> **Warning:**
>
> If you already have a VPN connecting to your VPC or plan to set up one, it is strongly recommended that you go to the `spec.grafana.service.annotations` section in the `db-monitor.yaml` file and set `service.beta.kubernetes.io/alicloud-loadbalancer-address-type` to `intranet` for security.

4.1.4.6 Upgrade

To upgrade the TiDB cluster, modify the `spec.version` variable by executing `kubectl --kubeconfig credentials/kubeconfig edit tc ${tidb_cluster_name} -n ↪ ${namespace}`.

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

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

4.1.4.7 Scale out the TiDB cluster

To scale out the TiDB cluster, modify `tikv_count`, `tiflash_count`, `cdc_count`, or `tidb_count` in the `terraform.tfvars` file, and then run `terraform apply` to scale out the number of nodes for the corresponding components.

After the nodes scale out, modify the `replicas` of the corresponding components by running `kubectl --kubeconfig credentials/kubeconfig edit tc ${tidb_cluster_name ↪ } -n ${namespace}`.

Note:

- Because it is impossible to determine which node will be taken offline during the scale-in process, the scale-in of TiDB clusters is currently not supported.
- The scale-out process takes a few minutes. You can watch the status by running `kubectl --kubeconfig credentials/kubeconfig get po ↪ -n ${namespace} --watch`.

4.1.4.8 Configure

4.1.4.8.1 Configure TiDB Operator

You can set the variables in `terraform.tfvars` to configure TiDB Operator. Most configuration items can be modified after you understand the semantics based on the comments of the `variable`. 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.

4.1.4.8.2 Configure the TiDB cluster

See TiDB Operator API Documentation and Configure a TiDB Cluster.

4.1.4.9 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
}

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
}
```

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`
`cluster_name`	The TiDB cluster name (required and unique)	`nil`
`tidb_version`	The TiDB cluster version	`v3.0.1`

Parameter	Description	Default value
tidb_cluster_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
tiflash_count	The count of TiFlash nodes	2
tiflash_instance_type	The TiFlash instance type	ecs.i2.2xlarge
cdc_count	The count of TiCDC nodes	3

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

Parameter	Description	Default value
`override_values`	The values 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"]`

Parameter	Description	Default value
`create_tidb_cluster_release`	Whether to create the TiDB cluster using Helm	`false`

4.1.4.10 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    = ${REGION}
    access_key = ${ACCESS_KEY}
    secret_key = ${SECRET_KEY}
}

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

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

provider "helm" {
```

```
    alias   = "default"
    insecure = true
    install_tiller = false
    kubernetes {
        config_path = module.tidb-operator.kubeconfig_filename
    }
}

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

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

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

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

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.

4.1.4.11 Destroy

1. Refer to Destroy a TiDB cluster to delete the cluster.

2. Destroy the ACK cluster by running the following command:

```
terraform destroy
```

If the Kubernetes cluster is not successfully created, the `destroy` operation might return an error and fail. In such cases, manually remove the Kubernetes resources from the local state:

```
terraform state list
```

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

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

> **Note:**
>
> You have to manually delete the cloud disk used by the components in the Alibaba Cloud console.

4.1.4.12 Limitation

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

4.1.5 In Self-managed Kubernetes

4.1.5.1 Prerequisites for TiDB in Kubernetes

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

4.1.5.1.1 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
Helm	v3.0.0+

4.1.5.1.2 Configure the firewall

It is recommended that you disable the firewall.

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

If you cannot stop the firewalld service, to ensure the normal operation of Kubernetes, take the following steps:

1. Enable the following ports on the master, and then restart the service:

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

# Set it when you need to expose NodePort on the master node.
firewall-cmd --permanent --add-port=30000-32767/tcp
systemctl restart firewalld
```

2. Enable the following ports on the nodes, and then restart the service:

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

systemctl restart firewalld
```

4.1.5.1.3 Configure Iptables

The FORWARD chain is configured to `ACCEPT` by default and is set in the startup script:

```
iptables -P FORWARD ACCEPT
```

4.1.5.1.4 Disable SELinux

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

4.1.5.1.5 Disable swap

To make kubelet work, you need to turn off swap and comment out the swap-related line in the `/etc/fstab` file.

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

4.1.5.1.6 Configure kernel parameters

Configure the kernel parameters as follows. You can also adjust them according to your environment:

```
modprobe br_netfilter

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

sysctl --system
```

4.1.5.1.7 Configure the Irqbalance service

The Irqbalance service binds the interrupts of each equipment to different CPUs respectively. This avoids the performance bottleneck when all interrupt requests are sent to the same CPU.

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

4.1.5.1.8 Configure the CPUfreq governor mode

To make full use of CPU performance, set the CPUfreq governor mode to `performance`. For details, see Configure the CPUfreq governor mode on the target machine.

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

4.1.5.1.9 Configure `ulimit`

The TiDB cluster uses many file descriptors by default. The `ulimit` of the worker node must be greater than or equal to `1048576`.

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

4.1.5.1.10 Docker service

It is recommended to install Docker CE 18.09.6 or later versions. See Install Docker for details.

After the installation, take the following steps:

1. Save the Docker data to a separate disk. The data mainly contains images and the container logs. To implement this, set the `--data-root` parameter:

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

   The above command sets the data directory of Docker to `/data1/docker`.

2. Set `ulimit` for the Docker daemon:

   1. Create the systemd drop-in directory for the docker service:

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

2. Create a file named as `/etc/systemd/system/docker.service.d/limit-nofile.conf`, and configure the value of the `LimitNOFILE` parameter. The value must be a number equal to or greater than `1048576`.

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

> **Note:**
> DO NOT set the value of `LimitNOFILE` to `infinity`. Due to a bug of `systemd`, the `infinity` value of `systemd` in some versions is `65536`.

3. Reload the configuration.

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

4.1.5.1.11 Kubernetes service

To deploy a multi-master, highly available cluster, see Kubernetes documentation.

The configuration of the Kubernetes master depends on the number of nodes. More nodes consumes more resources. You can adjust the number of nodes as needed.

Nodes in a Kubernetes cluster	Kubernetes master configuration
1-5	1vCPUs 4GB Memory
6-10	2vCPUs 8GB Memory
11-100	4vCPUs 16GB Memory
101-250	8vCPUs 32GB Memory
251-500	16vCPUs 64GB Memory
501-5000	32vCPUs 128GB Memory

After Kubelet is installed, take the following steps:

1. Save the Kubelet data to a separate disk (it can share the same disk with Docker). The data mainly contains the data used by emptyDir. To implement this, set the `--root-dir` parameter:

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

The above command sets the data directory of Kubelet to `/data1/kubelet`.

2. Reserve compute resources by using Kubelet, to ensure that the system process of the machine and the kernel process of Kubernetes have enough resources for operation in heavy workloads. This maintains the stability of the entire system.

4.1.5.1.12 TiDB cluster's requirements for resources

To determine the machine configuration, see Server recommendations.

In a production environment, avoid deploying TiDB instances on a kubernetes master, or deploy as few TiDB instances as possible. 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.

4.1.5.2 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:**
>
> Do not delete a PV in any case unless you are familiar with the underlying volume provisioner. Deleting a PV manually can cause orphaned volumes and unexpected behavior.

4.1.5.2.1 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.

4.1.5.2.2 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} -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}
   ```

4.1.5.2.3 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.

Step 1: Pre-allocate local storage

- For a disk that stores TiKV data, you can mount the disk into the `/mnt/ssd` directory.

  To achieve high performance, it is recommanded to allocate TiDB a dedicated disk, and the recommended disk type is SSD.

- For a disk that stores PD data, follow the steps to mount the disk. First, create multiple directories in the disk, and bind mount the directories into the `/mnt/sharedssd` directory.

> **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 that stores monitoring data, follow the steps to mount the disk. First, create multiple directories in the disk, and bind mount the directories into the `/mnt/monitoring` directory.

> **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 that stores TiDB Binlog and backup data, follow the steps to mount the disk. First, create multiple directories in the disk, and bind mount the directories them into the `/mnt/backup` directory.

> **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. All Ad-hoc full backup tasks and all scheduled full backup tasks share one PV.

The `/mnt/ssd`, `/mnt/sharedssd`, `/mnt/monitoring`, and `/mnt/backup` directories mentioned above are discovery directories used by local-volume-provisioner. local-volume-provisioner creates a corresponding PV for each subdirectory in discovery directory.

Step 2: Deploy local-volume-provisioner

Online deployment

1. Download the deployment file for local-volume-provisioner.

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

2. If you use the same discovery directory as described in Step 1: Pre-allocate local storage, you can skip this step. If you use a different path of discovery directory than in the previous step, you need to modify the ConfigMap and DaemonSet spec.

   - Modify the `data.storageClassMap` field in the ConfigMap spec:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: local-provisioner-config
  namespace: kube-system
data:
  # ...
  storageClassMap: |
    ssd-storage:
      hostDir: /mnt/ssd
      mountDir: /mnt/ssd
    shared-ssd-storage:
      hostDir: /mnt/sharedssd
      mountDir: /mnt/sharedssd
    monitoring-storage:
      hostDir: /mnt/monitoring
      mountDir: /mnt/monitoring
    backup-storage:
      hostDir: /mnt/backup
      mountDir: /mnt/backup
```

     For more configuration about local-volume-provisioner, refer to Configuration.

   - Modify `volumes` and `volumeMounts` fields in the DaemonSet spec to ensure the discovery directory can be mounted to the corresponding directory in the Pod:

```
......
      volumeMounts:
        - mountPath: /mnt/ssd
```

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

3. Deploy `local-volume-provisioner`.

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

4. Check status of Pod and PV.

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

`local-volume-provisioner` creates a PV for each mounting point under the discovery directory.

> **Note:**
> If no mount point is in the discovery directory, no PV is created and the output is empty.

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

Offline deployment

Steps of offline deployment is same as online deployment, except the following:

- Download the `local-volume-provisioner.yaml` file on a machine with Internet access, then upload it to the server and install it.
- `local-volume-provisioner` is a DaemonSet that starts a Pod on every Kubernetes worker node. The Pod uses the `quay.io/external_storage/local-volume-provisioner:v2.3.4` image. If the server does not have access to the Internet, download this Docker image on a machine with Internet access:

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

Copy the `local-volume-provisioner-v2.3.4.tar` file to the server, and execute the `docker load` command to load the file on the server:

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

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.

For more information on local PV in Kubernetes, refer to Best Practices.

4.1.5.2.4 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`.

Delete PV and data

When the reclaim policy of PVs is set to `Retain`, if you have confirmed that the data of a PV can be deleted, you can delete this PV and the corresponding data by strictly taking 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.

4.1.5.3 Deploy TiDB Operator in Kubernetes

This document describes how to deploy TiDB Operator in Kubernetes.

4.1.5.3.1 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 3

Deploy the Kubernetes cluster

TiDB Operator runs in the 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 want to deploy a very simple Kubernetes cluster for testing purposes, consult the Get Started document.

For some public cloud environments, refer to the following documents:

- Deploy on AWS EKS
- Deploy on GCP GKE
- Deploy on Alibaba Cloud ACK

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

It is recommended to enable RBAC in the Kubernetes cluster.

Install Helm

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

4.1.5.3.2 Configure local persistent volumes

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

4.1.5.3.3 Deploy TiDB Operator

Create CRD

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

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

If the server cannot access the Internet, you need to download the `crd.yaml` file on a machine with Internet access before installing:

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

If the following message is displayed, the CRD installation is successful:

```
kubectl get crd
```

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

Customize TiDB Operator deployment

To deploy TiDB Operator quickly, you can refer to Deploy TiDB Operator. This section describes how to customize the deployment of TiDB Operator.

After creating CRDs in the step above, there are two methods to deploy TiDB Operator on your Kubernetes cluster: online and offline.

Online deployment

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

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

> **Note:**
>
> `${chart_version}` represents the chart version of TiDB Operator. For example, `v1.1.15`. You can view the currently supported versions by running the `helm search repo -l tidb-operator` command.

2. Configure TiDB Operator

   TiDB Operator will use the `k8s.gcr.io/kube-scheduler` image. If you cannot download the image, you can modify the `scheduler.kubeSchedulerImageName` in the `${HOME}/tidb-operator/values-tidb-operator.yaml` file to `registry.cn-hangzhou.aliyuncs.com/google_containers/kube-scheduler`.

   You can modify other items such as `limits`, `requests`, and `replicas` as needed.

3. Deploy TiDB Operator

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

> **Note:**
>
> If the corresponding `tidb-admin` namespace does not exist, you can create the namespace first by running the `kubectl create namespace tidb-admin` command.

4. Upgrade TiDB Operator

   If you need to upgrade the TiDB Operator, modify the `${HOME}/tidb-operator/values-tidb-operator.yaml` file, and then execute the following command to upgrade:

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

Offline installation

If your server cannot access the Internet, install TiDB Operator offline by the following steps:

1. Download the `tidb-operator` chart

   If the server has no access to the Internet, you cannot configure the Helm repository to install the TiDB Operator component and other applications. At this time, you need

to download the chart file needed for cluster installation on a machine with Internet access, and then copy it to the server.

Use the following command to download the `tidb-operator` chart file:

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

Copy the `tidb-operator-v1.1.15.tgz` file to the target server and extract it to the current directory:

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

2. Download the Docker images used by TiDB Operator

   If the server has no access to the Internet, you need to download all Docker images used by TiDB Operator on a machine with Internet access and upload them to the server, and then use `docker load` to install the Docker image on the server.

   The Docker images used by TiDB Operator are:

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

   Among them, `k8s.gcr.io/kube-scheduler:v1.16.9` should be consistent with the version of your Kubernetes cluster. You do not need to download it separately.

   Next, download all these images using the following command:

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

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

   Next, upload these Docker images to the server, and execute `docker load` to install these Docker images on the server:

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

3. Configure TiDB Operator

   TiDB Operator embeds a `kube-scheduler` to implement a custom scheduler. To configure the Docker image's name and version of this built-in `kube-scheduler` component, modify the `./tidb-operator/values.yaml` file. For example, if `kube-scheduler` in your Kubernetes cluster uses the image `k8s.gcr.io/kube-scheduler:v1.16.9`, set `./tidb-operator/values.yaml` as follows:

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

   You can modify other items such as `limits`, `requests`, and `replicas` as needed.

4. Install TiDB Operator

   Install TiDB Operator using the following command:

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

> **Note:**
>
> If the corresponding `tidb-admin` namespace does not exist, you can create the namespace first by running the `kubectl create namespace` ↪ `tidb-admin` command.

5. Upgrade TiDB Operator

   If you need to upgrade TiDB Operator, modify the `./tidb-operator/values.yaml` file, and then execute the following command to upgrade:

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

4.1.5.3.4 Customize TiDB Operator

To customize TiDB Operator, modify `${HOME}/tidb-operator/values-tidb-operator.yaml`. The rest sections of the document use `values.yaml` to refer to `${HOME}/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}
    ↪ --namespace=tidb-admin -f ${HOME}/tidb-operator/values-tidb-operator.
    ↪ yaml
```

4.1.5.4 Configure a TiDB Cluster in Kubernetes

This document introduces how to configure a TiDB cluster for production deployment. It covers the following content:

- Configure resources
- Configure TiDB deployment
- Configure high availability

4.1.5.4.1 Configure resources

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, you need to configure resources of these components according to their needs. For details, refer to 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 quality of service (QoS) by making `limits` equal to `requests` when configuring resources. For details, refer to 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`, apart from setting Guaranteed-level QoS as mentioned above. For details, refer to Control CPU Management Policies on the Node.

4.1.5.4.2 Configure TiDB deployment

To configure a TiDB deployment, you need to configure the `TiDBCluster` CR. Refer to the TidbCluster example for an example. For the complete configurations of `TiDBCluster` CR, refer to API documentation.

> **Note:**
>
> It is recommended to organize configurations for a TiDB cluster under a directory of `cluster_name` and save it as `${cluster_name}/tidb-cluster.yaml`. The modified configuration is not automatically applied to the TiDB cluster by default. The new configuration file is loaded only when the Pod restarts.

Cluster name

The cluster name can be configured by changing `metadata.name` in the `TiDBCuster` CR.

Version

Usually, components in a cluster are in the same version. It is recommended to configure `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.baseImage` and `spec.version`, if you need to configure different versions for different components, you can configure `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`.

Here are the formats of the parameters:

- `spec.version`: the format is `imageTag`, such as `v5.0.6`
- `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.baseImage`: the format is `imageName`, such as `pingcap/tidb`
- `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`: the format is `imageTag`, such as `v5.0.6`

Recommended configuration

configUpdateStrategy

It is recommended that you configure `spec.configUpdateStrategy: RollingUpdate` to enable automatic update of configurations. This way, every time the configuration is updated, all components are rolling updated automatically, and the modified configuration is applied to the cluster.

enableDynamicConfiguration

It is recommended that you configure `spec.enableDynamicConfiguration: true` to enable the `--advertise-status-addr` startup parameter for TiKV.

Versions required:

- TiDB 4.0.1 or later versions
- TiDB Operator 1.1.1 or later versions

pvReclaimPolicy

It is recommended that you configure `spec.pvReclaimPolicy: Retain` to ensure that the PV is retained even if the PVC is deleted. This is to ensure your data safety.

mountClusterClientSecret

PD and TiKV supports configuring `mountClusterClientSecret`. If TLS is enabled between cluster components, it is recommended to configure `spec.pd.mountClusterClientSecret: true` and `spec.tikv.mountClusterClientSecret: true`. Under such configuration, TiDB Operator automatically mounts the `${cluster_name}-cluster-client-secret` certificate to the PD and TiKV container, so you can conveniently use `pd-ctl` and `tikv-ctl`.

Storage

Storage Class

You can set the storage class by modifying `storageClassName` of each component in `${cluster_name}/tidb-cluster.yaml` and `${cluster_name}/tidb-monitor.yaml`. For the storage classes supported by the Kubernetes cluster, check with your system administrator.

Different components of a TiDB cluster have different disk requirements. Before deploying a TiDB cluster, refer to the Storage Configuration document to select an appropriate storage class for each component according to the storage classes supported by the current Kubernetes cluster and usage scenario.

> **Note:**
>
> When you create the TiDB cluster, if you set a storage class that does not exist in the Kubernetes cluster, then the TiDB cluster creation goes to the Pending state. In this situation, you must destroy the TiDB cluster in Kubernetes and retry the creation.

Multiple disks mounting

TiDB Operator supports mounting multiple PVs for PD, TiDB, TiKV, and TiCDC, which can be used for data writing for different purposes.

You can configure the `storageVolumes` field for each component to describe multiple user-customized PVs.

The meanings of the related fields are as follows:

- `storageVolume.name`: The name of the PV.
- `storageVolume.storageClassName`: The StorageClass that the PV uses. If not configured, `spec.pd/tidb/tikv.storageClassName` will be used.
- `storageVolume.storageSize`: The storage size of the requested PV.
- `storageVolume.mountPath`: The path of the container to mount the PV to.

For example:

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

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

Note:

TiDB Operator uses some mount paths by default. For example, it mounts `EmptyDir` to the `/var/log/tidb` directory for the TiDB Pod. Therefore, avoid duplicate `mountPath` when you configure `storageVolumes`.

HostNetwork

For PD, TiKV, TiDB, TiFlash, TiCDC, and Pump, you can configure the Pods to use the host namespace `HostNetwork`.

To enable `HostNetwork` for all supported components, configure `spec.hostNetwork: true`.

To enable `HostNetwork` for specified components, configure `hostNetwork: true` for the components.

Discovery

TiDB Operator starts a Discovery service for each TiDB cluster. The Discovery service can return the corresponding startup parameters for each PD Pod to support the startup of the PD cluster. You can configure resources of the Discovery service using `spec.discovery`. For details, see Managing Resources for Containers.

A `spec.discovery` configuration example is as follows:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v4.0.10
  pvReclaimPolicy: Retain
  discovery:
    limits:
      cpu: "0.2"
    requests:
      cpu: "0.2"
  pd:
    baseImage: pingcap/pd
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
...
```

Cluster topology

PD/TiKV/TiDB

The deployed cluster topology by default has three PD Pods, three TiKV Pods, and two TiDB Pods. In this deployment topology, the scheduler extender of TiDB Operator requires at least three nodes in the Kubernetes cluster to provide high availability. You can modify the `replicas` configuration to change the number of pods for each component.

> **Note:**
>
> If the number of Kubernetes cluster nodes is less than three, one 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 three, to start the TiDB cluster, you can reduce the number of PD Pods in the default deployment to `1`.

Enable TiFlash

If you want to enable TiFlash in the cluster, configure `spec.pd.config.replication.enable-placement-rules: true` and configure `spec.tiflash` in the `${cluster_name}/tidb-cluster.yaml` file as follows:

```
pd:
  config: |
    ...
    [replication]
    enable-placement-rules = true
tiflash:
  baseImage: pingcap/tiflash
  maxFailoverCount: 3
  replicas: 1
  storageClaims:
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
```

TiFlash supports mounting multiple Persistent Volumes (PVs). If you want to configure multiple PVs for TiFlash, configure multiple `resources` in `tiflash.storageClaims`, each `resources` with a separate `storage request` and `storageClassName`. For example:

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

TiFlash mounts all PVs to directories such as `/data0` and `/data1` in the container in the order of configuration. TiFlash has four log files. The proxy log is printed in the standard output of the container. The other three logs are stored in the disk under the `/data0` ↪ directory by default, which are `/data0/logs/flash_cluster_manager.log`, `/ data0/` ↪ `logs/error.log`, `/data0/logs/server.log`. To modify the log storage path, refer to Configure TiFlash parameters.

> **Warning:**
>
> Since TiDB Operator will mount PVs automatically in the **order** of the items in the `storageClaims` list, if you need to add more disks to TiFlash, make

> sure to append the new item only to the **end** of the original items, and **DO NOT** modify the order of the original items.

Enable TiCDC

If you want to enable TiCDC in the cluster, you can add TiCDC spec to the `TiDBCluster` CR. For example:

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

Configure TiDB components

This section introduces how to configure the parameters of TiDB/TiKV/PD/TiFlash/TiCDC.

Configure TiDB parameters

TiDB parameters can be configured by `spec.tidb.config` in TidbCluster Custom Resource.

For example:

For TiDB Operator v1.1.6 and later versions, configure the parameters in the TOML format as follows:

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

For TiDB Operator versions earlier than v1.1.6, configure the parameters in the YAML format as follows:

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

For all the configurable parameters of TiDB, refer to TiDB Configuration File.

> **Note:**
>
> If you deploy your TiDB cluster using CR, make sure that `Config: {}` is set, no matter you want to modify `config` or not. Otherwise, TiDB components might not be started successfully. This step is meant to be compatible with `Helm` deployment.

Configure TiKV parameters

TiKV parameters can be configured by `spec.tikv.config` in TidbCluster Custom Resource.

For example:

For TiDB Operator v1.1.6 and later versions, configure the parameters in the TOML format as follows:

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

```
    image: pingcap/tikv:v5.0.1
    config: |
      [storage]
        [storage.block-cache]
          capacity = "16GB"
    replicas: 1
    requests:
      cpu: 2
```

For TiDB Operator versions earlier than v1.1.6, configure the parameters in the YAML format as follows:

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

For all the configurable parameters of TiKV, refer to TiKV Configuration File.

> **Note:**
>
> If you deploy your TiDB cluster using CR, make sure that `Config: {}` is set, no matter you want to modify `config` or not. Otherwise, TiKV components might not be started successfully. This step is meant to be compatible with `Helm` deployment.

Configure PD parameters

PD parameters can be configured by `spec.pd.config` in TidbCluster Custom Resource.

For example:

For TiDB Operator v1.1.6 and later versions, configure the parameters in the TOML format as follows:

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

For TiDB Operator versions earlier than v1.1.6, configure the parameters in the YAML format as follows:

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

For all the configurable parameters of PD, refer to PD Configuration File.

Note:

- If you deploy your TiDB cluster using CR, make sure that `Config: {}` is set, no matter you want to modify `config` or not. Otherwise, PD components might not be started successfully. This step is meant to be compatible with `Helm` deployment.
- After the cluster is started for the first time, some PD configuration items are persisted in etcd. The persisted configuration in etcd takes precedence over that in PD. Therefore, after the first start, you cannot modify some PD configuration using parameters. You need to dynamically modify the configuration using SQL statements, pd-ctl, or PD server API. Currently, among all the configuration items listed in Modify PD configuration online, except `log.level`, all the other configuration items cannot be modified using parameters after the first start.

Configure TiFlash parameters

TiFlash parameters can be configured by `spec.tiflash.config` in TidbCluster Custom Resource.

For example:

For TiDB Operator v1.1.6 and later versions, configure the parameters in the TOML format as follows:

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

For TiDB Operator versions earlier than v1.1.6, configure the parameters in the YAML format as follows:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  ...
  tiflash:
    config:
      config:
       flash:
          flash_cluster:
            log: "/data0/logs/flash_cluster_manager.log"
        logger:
          count: 10
          level: information
          errorlog: "/data0/logs/error.log"
          log: "/data0/logs/server.log"
```

For all the configurable parameters of TiFlash, refer to TiFlash Configuration File.

Configure TiCDC start parameters

You can configure TiCDC start parameters through `spec.ticdc.config` in TidbCluster Custom Resource.

For example:

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

For all configurable start parameters of TiCDC, see TiCDC configuration.

Configure automatic failover thresholds of PD, TiDB, TiKV, and TiFlash

The automatic failover feature is enabled by default in TiDB Operator. When the Pods of PD, TiDB, TiKV, TiFlash fail or the corresponding nodes fail, TiDB Operator performs failover automatically and replenish the number of Pod replicas by scaling the corresponding components.

To avoid that the automatic failover feature creates too many Pods, you can configure the threshold of the maximum number of Pods that TiDB Operator can create during failover for each component. The default threshold is `3`. If the threshold for a component is configured to `0`, it means that the automatic failover feature is disabled for this component. An example configuration is as follows:

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

Configure graceful upgrade for TiDB cluster

When you perform a rolling update to the TiDB cluster, Kubernetes sends a `TERM` signal to the TiDB server before it stops the TiDB Pod. When the TiDB server receives the `TERM`

signal, it tries to wait for all connections to close. After 15 seconds, the TiDB server forcibly closes all the connections and exits the process.

Starting from v1.1.2, TiDB Operator supports gracefully upgrading the TiDB cluster. You can enable this feature by configuring the following items:

- `spec.tidb.terminationGracePeriodSeconds`: The longest tolerable duration to delete the old TiDB Pod during the rolling upgrade. If this duration is exceeded, the TiDB Pod will be deleted forcibly.
- `spec.tidb.lifecycle`: Sets the `preStop` hook for the TiDB Pod, which is the operation executed before the TiDB server stops.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v5.0.6
  pvReclaimPolicy: Retain
  discovery: {}
  pd:
    baseImage: pingcap/pd
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    service:
      type: ClusterIP
    config: {}
    terminationGracePeriodSeconds: 60
    lifecycle:
      preStop:
        exec:
          command:
          - /bin/sh
          - -c
```

```
        - "sleep 10 && kill -QUIT 1"
```

The YAML file above:

- Sets the longest tolerable duration to delete the TiDB Pod to 60 seconds. If the client does not close the connections after 60 seconds, these connections will be closed forcibly. You can adjust the value according to your needs.
- Sets the value of `preStop` hook to `sleep 10 && kill -QUIT 1`. Here `PID 1` refers to the PID of the TiDB server process in the TiDB Pod. When the TiDB server process receives the signal, it exits only after all the connections are closed by the client.

When Kubernetes deletes the TiDB Pod, it also removes the TiDB node from the service endpoints. This is to ensure that the new connection is not established to this TiDB node. However, because this process is asynchronous, you can make the system sleep for a few seconds before you send the `kill` signal, which makes sure that the TiDB node is removed from the endpoints.

Configure graceful upgrade for TiKV cluster

During TiKV upgrade, TiDB Operator evicts all Region leaders from TiKV Pod before restarting TiKV Pod. Only after the eviction is completed (which means the number of Region leaders on TiKV Pod drops to 0) or the eviction exceeds the specified timeout (10 minutes by default), TiKV Pod is restarted.

If the eviction of Region leaders exceeds the specified timeout, restarting TiKV Pod causes issues such as failures of some requests or more latency. To avoid the issues, you can configure the timeout `spec.tikv.evictLeaderTimeout` (10 minutes by default) to a larger value. For example:

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

Warning:

If the TiKV version is earlier than 4.0.14 or 5.0.3, due to a bug of TiKV, you need to configure the timeout `spec.tikv.evictLeaderTimeout` as large as possible to ensure that all Region leaders on the TiKV Pod can be evicted within the timeout. If you are not sure about the proper value, greater than '1500m' is recommended.

Configure PV for TiDB slow logs

By default, TiDB Operator creates a `slowlog` volume (which is an `EmptyDir`) to store the slow logs, mounts the `slowlog` volume to `/var/log/tidb`, and prints slow logs in the `stdout` through a sidecar container.

> **Warning:**
>
> By default, after a Pod is deleted (for example, rolling update), the slow query logs stored using the `EmptyDir` volume are lost. Make sure that a log collection solution has been deployed in the Kubernetes cluster to collect logs of all containers. If you do not deploy such a log collection solution, you **must** make the following configuration to use a persistent volume to store the slow query logs.

If you want to use a separate PV to store the slow logs, you can specify the name of the PV in `spec.tidb.slowLogVolumeName`, and then configure the PV in `spec.tidb.storageVolumes` or `spec.tidb.additionalVolumes`.

This section shows how to configure PV using `spec.tidb.storageVolumes` or `spec.tidb.additionalVolumes`.

Configure using `spec.tidb.storageVolumes`

Configure the `TidbCluster` CR as the following example. In the example, TiDB Operator uses the `${volumeName}` PV to store slow logs. The log file path is `${mountPath}/${volumeName}`.

For how to configure the `spec.tidb.storageVolumes` field, refer to Multiple disks mounting.

> **Warning:**
>
> You need to configure `storageVolumes` before creating the cluster. After the cluster is created, adding or removing `storageVolumes` is no longer supported. For the `storageVolumes` already configured, except for increasing `storageVolume.storageSize`, other modifications are not supported. To increase `storageVolume.storageSize`, you need to make sure that the corresponding StorageClass supports dynamic expansion.

```
tidb:
  ...
  separateSlowLog: true # can be ignored
  slowLogVolumeName: ${volumeName}
```

```
  storageVolumes:
    # name must be consistent with slowLogVolumeName
    - name: ${volumeName}
      storageClassName: ${storageClass}
      storageSize: "1Gi"
      mountPath: ${mountPath}
```

Configure using `spec.tidb.additionalVolumes` (supported starting from v1.1.8)

In the following example, NFS is used as the storage, and TiDB Operator uses the `${volumeName}` PV to store slow logs. The log file path is `${mountPath}/${volumeName}`.

For the supported PV types, refer to Persistent Volumes.

```
tidb:
  ...
  separateSlowLog: true # can be ignored
  slowLogVolumeName: ${volumeName}
  additionalVolumes:
  # name must be consistent with slowLogVolumeName
  - name: ${volumeName}
    nfs:
      server: 192.168.0.2
      path: /nfs
  additionalVolumeMounts:
  # name must be consistent with slowLogVolumeName
  - name: ${volumeName}
    mountPath: ${mountPath}
```

Configure TiDB service

You need to configure `spec.tidb.service` so that TiDB Operator creates a service for TiDB. You can configure Service with different types according to the scenarios, such as `ClusterIP`, `NodePort`, `LoadBalancer`, etc.

General configurations

Different types of services share some general configurations as follows:

- `spec.tidb.service.annotations`: the annotation added to the Service resource.
- `spec.tidb.service.labels`: the labels added to the Service resource.

ClusterIP

`ClusterIP` exposes services through the internal IP of the cluster. When selecting this type of service, you can only access it within the cluster using ClusterIP or the Service domain name (`${cluster_name}-tidb.${namespace}`).

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

NodePort

If there is no LoadBalancer, you can choose to expose the service through NodePort. NodePort exposes services through the node's IP and static port. You can access a NodePort service from outside of the cluster by requesting `NodeIP + NodePort`.

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

NodePort has two modes:

- `externalTrafficPolicy=Cluster`: All machines in the cluster allocate a NodePort port to TiDB, which is the default value.

  When using the `Cluster` mode, you can access the TiDB service through the IP and NodePort of any machine. If there is no TiDB Pod on the machine, the corresponding request will be forwarded to the machine with TiDB Pod.

> **Note:**
>
> In this mode, the request source IP obtained by the TiDB service is the host IP, not the real client source IP, so access control based on the client source IP is not available in this mode.

`-externalTrafficPolicy=Local`: Only the machine that TiDB is running on allocates a NodePort port to access the local TiDB instance.

LoadBalancer

If the TiDB cluster runs in an environment with LoadBalancer, such as on GCP or AWS, it is recommended to use the LoadBalancer feature of these cloud platforms by setting `tidb.service.type=LoadBalancer`.

```
spec:
  ...
  tidb:
```

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

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

4.1.5.4.3 Configure high availability

> **Note:**
>
> TiDB Operator provides a custom scheduler that guarantees TiDB service can tolerate host-level failures through the specified scheduling algorithm. Currently, the TiDB cluster uses this scheduler as the default scheduler, which is configured through the item `spec.schedulerName`. This section focuses on configuring a TiDB cluster to tolerate failures at other levels such as rack, zone, or region. This section is optional.

TiDB is a distributed database and its high availability 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 high availability are described separately as follows.

High availability of TiDB service

Use nodeSelector to schedule Pods

By configuring the `nodeSelector` field of each component, you can specify the specific nodes that the component Pods are scheduled onto. For details on `nodeSelector`, refer to nodeSelector.

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

```
  tidb:
    nodeSelector:
      node-role.kubernetes.io/tidb: true
    # ...
```

Use tolerations to schedule Pods

By configuring the `tolerations` field of each component, you can allow the component Pods to schedule onto nodes with matching taints. For details on taints and tolerations, refer to Taints and Tolerations.

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

Use affinity to schedule Pods

By configuring `PodAntiAffinity`, you can avoid the situation in which different instances of the same component are deployed on the same physical topology node. In this way, disaster recovery (high availability) is achieved. For the user guide of Affinity, see Affinity & AntiAffinity.

The following is an example of a typical service high availability setup:

```
affinity:
 podAntiAffinity:
```

```
  preferredDuringSchedulingIgnoredDuringExecution:
  # this term works when the nodes have the label named region
  - weight: 10
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app.kubernetes.io/instance: ${cluster_name}
          app.kubernetes.io/component: "pd"
      topologyKey: "region"
      namespaces:
      - ${namespace}
  # this term works when the nodes have the label named zone
  - weight: 20
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app.kubernetes.io/instance: ${cluster_name}
          app.kubernetes.io/component: "pd"
      topologyKey: "zone"
      namespaces:
      - ${namespace}
  # this term works when the nodes have the label named rack
  - weight: 40
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app.kubernetes.io/instance: ${cluster_name}
          app.kubernetes.io/component: "pd"
      topologyKey: "rack"
      namespaces:
      - ${namespace}
  # this term works when the nodes have the label named kubernetes.io/
     ↪ hostname
  - weight: 80
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app.kubernetes.io/instance: ${cluster_name}
          app.kubernetes.io/component: "pd"
      topologyKey: "kubernetes.io/hostname"
      namespaces:
      - ${namespace}
```

High availability of data

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

To add the data high availability feature in Kubernetes:

1. Set the label collection of topological location for PD

   Replace the `location-labels` information in the `pd.config` with the label collection that describes the topological location 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 `host` in the `location-labels`, TiDB Operator will get the value from the `kubernetes.io/hostname` in the node label.

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

   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`.

4.1.5.5 Deploy TiDB in General Kubernetes

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

4.1.5.5.1 Prerequisites

- Meet prerequisites.
- Complete deploying TiDB Operator.
- Configure the TiDB cluster

4.1.5.5.2 Deploy the TiDB cluster

1. Create `Namespace`:

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

> **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 `cluster_name`.

2. Deploy the TiDB cluster:

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

> **Note:**
>
> It is recommended to organize configurations for a TiDB cluster under a directory of `cluster_name` and save it as `${cluster_name}/tidb-cluster.yaml`.

If the server does not have an external network, you need to download the Docker image used by the TiDB cluster on a machine with Internet access and upload it to the server, and then use `docker load` to install the Docker image on the server.

To deploy a TiDB cluster, you need the following Docker images (assuming the version of the TiDB cluster is v5.0.6):

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

Next, download all these images with the following command:

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

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

Next, upload these Docker images to the server, and execute `docker load` to install these Docker images on the server:

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

3. View the Pod status:

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

You can use TiDB Operator to deploy and manage multiple TiDB clusters in a single Kubernetes cluster by repeating the above procedure and replacing `cluster_name` with a different name.

Different clusters can be in the same or different `namespace`, which is based on your actual needs.

> **Note:**
>
> If you need to deploy a TiDB cluster on ARM64 machines, refer to Deploy a TiDB Cluster on ARM64 Machines.

4.1.5.5.3 Initialize the TiDB cluster

If you want to initialize your cluster after deployment, refer to Initialize a TiDB Cluster in Kubernetes.

> **Note:**
>
> By default, TiDB (starting from v4.0.2) periodically shares usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see Telemetry.

4.1.5.6 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:**
>
> - After creating the TiDB cluster, if you manually change the password of the `root` account, the initialization will fail.
> - The following steps apply only when you have created a cluster for the first time. Further configuration or modification after the initial cluster creation is not valid.

4.1.5.6.1 Configure TidbInitializer

Refer to TidbInitializer configuration example, API documentation, and the following steps to complete TidbInitializer Custom Resource (CR), and save it to the `${cluster_name}/tidb-initializer.yaml` file. When referring to the TidbInitializer configuration example and API documentation, you need to switch the branch to the TiDB Operator version currently in use.

Set the cluster namespace and name

In the `${cluster_name}/tidb-initializer.yaml` file, modify the `spec.cluster.namespace` and `spec.cluster.name` fields:

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

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 methods:

- Create a `secret` to specify the password for `root`:

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

- If you want to create more than one user, add the desired username and the password in the above command. For example:

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

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

4.1.5.6.2 Set a host that has access to TiDB

To set a host that has access to TiDB, modify the `permitHost: ${mysql_client_host_name}` configuration item in `${cluster_name}/tidb-initializer.yaml`. If it is not set, all hosts have access to TiDB. For details, refer to Mysql GRANT host name.

4.1.5.6.3 Initialize SQL statements in batch

The cluster can also automatically execute the SQL statements in batch in `initSql` during the initialization. This function can be used to create some databases or tables for the cluster and perform 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`:

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

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

4.1.5.6.4 Initialize the cluster

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

The above command automatically creates an initialized Job. This Job tries to set the initial password for the `root` account using the `secret` object provided. It also tries to create other accounts and passwords, if they are specified.

After the initialization, the Pod state becomes `Completed`. If you log in via MySQL client later, you need to specify the password created by the Job.

If the server does not have an external network, you need to download the Docker image used for cluster initialization on a machine with an external network and upload it to the server, and then use `docker load` to install the Docker image on the server.

The following Docker images are used to initialize a TiDB cluster:

```
tnir/mysqlclient:latest
```

Next, download all these images with the following command:

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

Next, upload these Docker images to the server, and execute `docker load` to install these Docker images on the server:

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

4.1.5.7 Access the TiDB Cluster

This document describes how to access the TiDB cluster.

You can configure Service with different types according to the scenarios, such as `ClusterIP`, `NodePort`, `LoadBalancer`, etc., and use different access methods for different types.

You can obtain TiDB Service information by running the following command:

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

For example:

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

The above example describes the information of the `basic-tidb` service in the `default` namespace. The type is `NodePort`, ClusterIP is `10.233.6.240`, ServicePort is `4000` and `10080`, and the corresponding NodePort is `32498` and `30171`.

> **Note:**
>
> The default authentication plugin of MySQL 8.0 is updated from `mysql_native_password` to `caching_sha2_password`. Therefore, if you use MySQL client from MySQL 8.0 to access the TiDB service (TiDB version earlier than v4.0.7), and if the user account has a password, you need to explicitly specify the `--default-auth=mysql_native_password` parameter.

4.1.5.7.1 ClusterIP

`ClusterIP` exposes services through the internal IP of the cluster. When selecting this type of service, you can only access it within the cluster by the following methods:

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

4.1.5.7.2 NodePort

If there is no LoadBalancer, you can choose to expose the service through NodePort. NodePort exposes services through the node's IP and static port. You can access a NodePort service from outside of the cluster by requesting `NodeIP + NodePort`.

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

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

4.1.5.7.3 LoadBalancer

If the TiDB cluster runs in an environment with LoadBalancer, such as on GCP or AWS, it is recommended to use the LoadBalancer feature of these cloud platforms by setting `tidb.service.type=LoadBalancer`.

To access TiDB Service through LoadBalancer, refer to EKS, GKE and ACK.

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

4.1.6 Deploy a TiDB Cluster on ARM64 Machines

This document describes how to deploy a TiDB cluster on ARM64 machines.

4.1.6.1 Prerequisites

Before starting the process, make sure that Kubernetes clusters are deployed on your ARM64 machines. If Kubernetes clusters are not deployed, refer to Deploy the Kubernetes cluster.

4.1.6.2 Deploy TiDB operator

The process of deploying TiDB operator on ARM64 machines is the same as the process of Deploy TiDB Operator in Kubernetes. The only difference is the following configuration in the step Customize TiDB operator deployment: after getting the `values.yaml` file of the `tidb-operator` chart, you need to modify the `operatorImage` and `tidbBackupManagerImage` fields in that file to the ARM64 image versions. For example:

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

4.1.6.3 Deploy a TiDB cluster

The process of deploying a TiDB cluster on ARM64 machines is the same as the process of Deploy TiDB in General Kubernetes. The only difference is that, in the TidbCluster definition file, you need to set the images of the related components to the ARM64 versions. For example:

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

```
    # ...
  tiflash:
    baseImage: pingcap/tiflash-arm64
    # ...
```

4.1.6.4 Initialize a TiDB cluster

The process of initializing a TiDB cluster on ARM64 machines is the same as the process of Initialize a TiDB Cluster in Kubernetes. The only difference is that you need to modify the `spec.image` field in the TidbInitializer definition file to the ARM64 image version. For example:

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

4.1.6.5 Deploy monitoring for a TiDB cluster

The process of deploying monitoring for a TiDB cluster on ARM64 machines is the same as the process of Deploy Monitoring and Alerts for a TiDB Cluster. The only difference is that you need to modify the `spec.initializer.baseImage` and `spec.reloader.baseImage` fields in the TidbMonitor definition file to the ARM64 image versions.

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

4.2 Deploy a Heterogeneous TiDB Cluster

This document describes how to deploy a heterogeneous cluster for an existing TiDB cluster.

4.2.1 Prerequisites

- You already have a TiDB cluster. If not, refer to Deploy TiDB in General Kubernetes.

4.2.2 Deploy a heterogeneous cluster

A heterogeneous cluster creates differentiated instances for an existing TiDB cluster. You can create a heterogeneous TiKV cluster with different configurations and labels to facilitate hotspot scheduling, or create a heterogeneous TiDB cluster for OLTP and OLAP workloads respectively.

4.2.2.1 Create a heterogeneous cluster

Save the following configuration as the `cluster.yaml` file. Replace `${heterogeneous_cluster_name}` with the desired name of your heterogeneous cluster, and replace `${origin_cluster_name}` with the name of the existing cluster.

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

```
    replicas: 1
    service:
      type: ClusterIP
    config: {}
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 1
    replicas: 1
    storageClaims:
      - resources:
          requests:
            storage: 1Gi
        storageClassName: standard
```

Execute the following command to create the heterogeneous cluster:

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

The configuration of a heterogeneous cluster is mostly the same as a normal TiDB cluster, except that it uses the `spec.cluster.name` field to join the target cluster.

4.2.2.2 Deploy the cluster monitoring component

Save the following configuration as the `tidbmonitor.yaml` file. Replace `${heterogeneous_cluster_name}` with the desired name of your heterogeneous cluster, and replace `${origin_cluster_name}` with the name of the existing cluster.

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

```
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

Execute the following command to create the heterogeneous cluster:

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

4.2.3 Deploy a TLS-enabled heterogeneous cluster

To enable TLS for a heterogeneous cluster, you need to explicitly declare the TLS configuration, issue the certificates using the same certification authority (CA) as the target cluster and create new secrets with the certificates.

If you want to issue the certificate using `cert-manager`, choose the same `Issuer` as that of the target cluster to create your `Certificate`.

For detailed procedures to create certificates for the heterogeneous cluster, refer to the following two documents:

- Enable TLS between TiDB Components
- Enable TLS for the MySQL Client

4.2.3.1 Create a TLS-enabled heterogeneous cluster

Save the following configuration as the `cluster.yaml` file. Replace `${heterogeneous_cluster_name` ↪ `}` with the desired name of your heterogeneous cluster, and replace `${origin_cluster_name` ↪ `}` with the name of the existing cluster.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${heterogeneous_cluster_name}
spec:
  tlsCluster:
    enabled: true
  configUpdateStrategy: RollingUpdate
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  discovery: {}
  cluster:
    name: ${origin_cluster_name}
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
```

```
  # if storageClassName is not set, the default Storage Class of the
    ↪ Kubernetes cluster will be used
  # storageClassName: local-storage
  requests:
    storage: "1Gi"
  config:
    storage:
      # In basic examples, we set this to avoid using too much storage.
      reserve-space: "0MB"
tidb:
  baseImage: pingcap/tidb
  replicas: 1
  service:
    type: ClusterIP
  config: {}
  tlsClient:
    enabled: true
tiflash:
  baseImage: pingcap/tiflash
  maxFailoverCount: 1
  replicas: 1
  storageClaims:
    - resources:
        requests:
          storage: 1Gi
      storageClassName: standard
```

- `spec.tlsCluster.enabled`: Determines whether to enable TLS between the components.
- `spec.tidb.tlsClient.enabled`: Determines whether to enable TLS for MySQL client.

Execute the following command to create the TLS-enabled heterogeneous cluster:

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

For the detailed configuration of a TLS-enabled heterogeneous cluster, see 'heterogeneous-tls' example.

4.3 Deploy TiFlash in Kubernetes

This document describes how to deploy TiFlash in Kubernetes.

4.3.1 Prerequisites

- Deploy TiDB Operator.

4.3.2 Fresh TiFlash deployment

To deploy TiFlash when deploying the TiDB cluster, refer to Deploy TiDB on General Kubernetes.

4.3.3 Add TiFlash to an existing TiDB cluster

Edit the `TidbCluster` Custom Resource:

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

Add the TiFlash configuration as follows:

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

To configure other parameters, refer to Configure a TiDB Cluster.

TiFlash supports mounting multiple Persistent Volumes (PVs). If you want to configure multiple PVs for TiFlash, configure multiple `resources` in `tiflash.storageClaims`, each `resources` with a separate `storage request` and `storageClassName`. For example:

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

> **Warning:**
>
> Since TiDB Operator will mount PVs automatically in the **order** of the items in the `storageClaims` list, if you need to add more disks to TiFlash, make sure to append the new item only to the **end** of the original items, and **DO NOT** modify the order of the original items.

TiDB Operator manages TiFlash by creating StatefulSet. Since `StatefulSet` does not support modifying `volumeClaimTemplates` after creation, updating `storageClaims` to add the disk cannot mount the additional PV to the Pod. There are two solutions:

- When deploying the TiFlash cluster for the first time, determine how many PVs are required and configure `storageClaims`.
- If you really want to add a PV, after configuring `storageClaims`, you need to manually delete the TiFlash StatefulSet (`kubectl delete sts -n ${namespace} ${cluster_name}-tiflash`) and wait for the TiDB Operator to recreate it.

To add TiFlash component to an existing TiDB cluster, you need to set `replication.enable-placement-rules: true` in PD. After you add the TiFlash configuration in `TidbCluster` by taking the above steps, TiDB Operator automatically configures `replication.enable-placement-rules: true` in PD.

If the server does not have an external network, refer to deploy the TiDB cluster to download the required Docker image on the machine with an external network and upload it to the server.

4.3.4 Remove TiFlash

1. Adjust the number of replicas of the tables replicated to the TiFlash cluster.

   To completely remove TiFlash, you need to set the number of replicas of all tables replicated to the TiFlash to `0`.

   1. To connect to the TiDB service, refer to the steps in Access the TiDB Cluster in Kubernetes.
   2. To adjust the number of replicas of the tables replicated to the TiFlash cluster, run the following command:

      ```
      alter table <db_name>.<table_name> set tiflash replica 0;
      ```

2. Wait for the TiFlash replicas of the related tables to be deleted.

   Connect to the TiDB service and run the following command. If you can not find the replication information of the related tables, it means that the replicas are deleted:

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA =
    ↪ '<db_name>' and TABLE_NAME = '<table_name>';
```

3. To remove TiFlash Pods, run the following command to modify `spec.tiflash.replicas` to 0:

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

4. Check the state of TiFlash Pods and TiFlash stores.

First, run the following command to check whether you delete the TiFlash Pod successfully:

```
```shell
kubectl get pod -n ${namespace} -l app.kubernetes.io/component=tiflash,app.
    ↪ kubernetes.io/instance=${cluster_name}
```
```

If the output is empty, it means that you delete the Pod of the TiFlash cluster successfully.

To check whether the stores of the TiFlash are in the `Tombstone` state, run the following command:

```
```shell
kubectl get tidbcluster ${cluster_name} -n ${namespace} -o yaml
```
```

The value of the `status.tiflash` field in the output result is similar to the example below.

```
```
tiflash:
    ...
    tombstoneStores:
    "88":
        id: "88"
        ip: basic-tiflash-0.basic-tiflash-peer.default.svc
        lastHeartbeatTime: "2020-12-31T04:42:12Z"
        lastTransitionTime: null
        leaderCount: 0
        podName: basic-tiflash-0
        state: Tombstone
```
```

```
    "89":
        id: "89"
        ip: basic-tiflash-1.basic-tiflash-peer.default.svc
        lastHeartbeatTime: "2020-12-31T04:41:50Z"
        lastTransitionTime: null
        leaderCount: 0
        podName: basic-tiflash-1
        state: Tombstone
```

Only after you delete all Pods of the TiFlash cluster successfully and all the TiFlash stores have changed to the `Tombstone` state, can you perform the next operation.

5. Delete the TiFlash StatefulSet.

To modify the TidbCluster CR and delete the `spec.tiflash` field, run the following command:

```shell
kubectl edit tidbcluster ${cluster_name} -n ${namespace}
```

To delete the TiFlash StatefulSet, run the following command:

```shell
kubectl delete statefulsets -n ${namespace} -l app.kubernetes.io/component=tiflash,app.kubernetes.io/instance=${cluster_name}
```

To check whether you delete the StatefulSet of the TiFlash cluster successfully, run the following command:

```shell
kubectl get sts -n ${namespace} -l app.kubernetes.io/component=tiflash,app.kubernetes.io/instance=${cluster_name}
```

If the output is empty, it means that you delete the StatefulSet of the TiFlash cluster successfully.

6. (Optional) Delete PVC and PV.

If you confirm that you do not use the data in TiFlash, and you want to delete the data, you need to strictly follow the steps below to delete the data in TiFlash:

1. Delete the PVC object corresponding to the PV

```
shell  kubectl delete pvc -n ${namespace} -l app.kubernetes.io/component
↪ =tiflash,app.kubernetes.io/instance=${cluster_name}
```

2. If the PV reclaim policy is `Retain`, the corresponding PV is still retained after you delete the PVC object. If you want to delete the PV, you can set the reclaim policy of the PV to `Delete`, and the PV can be deleted and recycled automatically.

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

In the above command, `${pv_name}` represents the PV name of the TiFlash cluster. You can check the PV name by running the following command:

```
shell  kubectl get pv -l app.kubernetes.io/component=tiflash,app.kubernetes
↪ .io/instance=${cluster_name}
```

4.3.5 Configuration notes for different versions

Starting from TiDB Operator v1.1.5, the default configuration of `spec.tiflash.config.config.flash.service_addr` is changed from `${clusterName}-tiflash-POD_NUM.${clusterName}-tiflash-peer.${namespace}.svc:3930` to `0.0.0.0:3930`, and TiFlash needs to configure `spec.tiflash.config.config.flash.service_addr` to `0.0.0.0:3930` since v4.0.5.

Therefore, for different TiFlash and TiDB Operator versions, you need to pay attention to the following configurations:

- If the TiDB Operator version <= v1.1.4
  - If the TiFlash version <= v4.0.4, no need to manually configure `spec.tiflash.config.config.flash.service_addr`.
  - If the TiFlash version >= v4.0.5, you need to set `spec.tiflash.config.config.flash.service_addr` to `0.0.0.0:3930` in the `TidbCluster` CR.
- If the TiDB Operator version >= v1.1.5
  - If the TiFlash version <= v4.0.4, you need to set `spec.tiflash.config.config.flash.service_addr` to `${clusterName}-tiflash-POD_NUM.${clusterName}-tiflash-peer.${namespace}.svc:3930` in the `TidbCluster` CR. `${clusterName}` and `${namespace}` need to be replaced according to the real case.
  - If the TiFlash version >= v4.0.5, no need to manually configure `spec.tiflash.config.config.flash.service_addr`.
  - If you upgrade from TiFlash v4.0.4 or lower versions to TiFlash v4.0.5 or higher versions, you need to delete the configuration of `spec.tiflash.config.config.flash.service_addr` in the `TidbCluster` CR.

4.4 Deploy TiCDC in Kubernetes

TiCDC is a tool for replicating the incremental data of TiDB. This document describes how to deploy TiCDC in Kubernetes using TiDB Operator.

You can deploy TiCDC when deploying a new TiDB cluster, or add the TiCDC component to an existing TiDB cluster.

4.4.1 Prerequisites

TiDB Operator is deployed.

4.4.2 Fresh TiCDC deployment

To deploy TiCDC when deploying the TiDB cluster, refer to Deploy TiDB in General Kubernetes.

4.4.3 Add TiCDC to an existing TiDB cluster

1. Edit `TidbCluster` Custom Resource:

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

2. Add the TiCDC configuration as follows:

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

3. After the deployment, enter a TiCDC Pod by running `kubectl exec`:

```
kubectl exec -it ${pod_name} -n ${namespace} -- sh
```

4. Manage the cluster and data replication tasks by using `cdc cli`.

```
/cdc cli capture list --pd=http://${cluster_name}-pd:2379
```

```
[
  {
    "id": "3ed24f6c-22cf-446f-9fe0-bf4a66d00f5b",
    "is-owner": false,
    "address": "${cluster_name}-ticdc-2.${cluster_name}-ticdc-peer.${
        ↪ namespace}.svc:8301"
  },
  {
```

```
    "id": "60e98ed7-cd49-45f4-b5ae-d3b85ba3cd96",
    "is-owner": false,
    "address": "${cluster_name}-ticdc-0.${cluster_name}-ticdc-peer.${
        ↪ namespace}.svc:8301"
  },
  {
    "id": "dc3592c0-dace-42a0-8afc-fb8506e8271c",
    "is-owner": true,
    "address": "${cluster_name}-ticdc-1.${cluster_name}-ticdc-peer.${
        ↪ namespace}.svc:8301"
  }
]
```

Starting from v4.0.3, TiCDC supports TLS. TiDB Operator supports enabling TLS for TiCDC since v1.1.3.

If TLS is enabled when you create the TiDB cluster, add TLS certificate-related parameters when you use `cdc cli`.

```
/cdc cli capture list --pd=https://${cluster_name}-pd:2379 --ca=/var/
    ↪ lib/cluster-client-tls/ca.crt --cert=/var/lib/cluster-client-tls/
    ↪ tls.crt --key=/var/lib/cluster-client-tls/tls.key
```

If the server does not have an external network, refer to deploy TiDB cluster to download the required Docker image on the machine with an external network and upload it to the server.

4.5 Deploy TiDB Binlog

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

4.5.1 Prerequisites

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

4.5.2 Deploy TiDB Binlog in 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, take the following steps.

4.5.2.1 Deploy Pump

1. Modify the `TidbCluster` CR file to add the Pump configuration.

   For example:

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

   Since v1.1.6, TiDB Operator supports passing raw TOML configuration to the component:

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

   Edit `version`, `replicas`, `storageClassName`, and `requests.storage` according to your cluster.

2. Set affinity and anti-affinity for TiDB and Pump.

   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, the affinity of TiDB and Pump 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 in TiDB components, 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. The steps are as follows:

- Configure `spec.tidb.affinity` as follows:

```
spec:
  tidb:
    affinity:
      podAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: "app.kubernetes.io/component"
                operator: In
                values:
                - "pump"
              - key: "app.kubernetes.io/managed-by"
                operator: In
                values:
                - "tidb-operator"
              - key: "app.kubernetes.io/name"
                operator: In
                values:
                - "tidb-cluster"
              - key: "app.kubernetes.io/instance"
                operator: In
                values:
                - ${cluster_name}
            topologyKey: kubernetes.io/hostname
```

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

```
spec:
  pump:
    affinity:
      podAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
```

```
          - key: "app.kubernetes.io/component"
            operator: In
            values:
            - "tidb"
          - key: "app.kubernetes.io/managed-by"
            operator: In
            values:
            - "tidb-operator"
          - key: "app.kubernetes.io/name"
            operator: In
            values:
            - "tidb-cluster"
          - key: "app.kubernetes.io/instance"
            operator: In
            values:
            - ${cluster_name}
        topologyKey: kubernetes.io/hostname
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: "app.kubernetes.io/component"
              operator: In
              values:
              - "pump"
            - key: "app.kubernetes.io/managed-by"
              operator: In
              values:
              - "tidb-operator"
            - key: "app.kubernetes.io/name"
              operator: In
              values:
              - "tidb-cluster"
            - key: "app.kubernetes.io/instance"
              operator: In
              values:
              - ${cluster_name}
          topologyKey: kubernetes.io/hostname
```

Note:

> If you update the affinity configuration of the TiDB components, it will cause rolling updates of the TiDB components in the cluster.

4.5.3 Deploy Drainer

To deploy multiple drainers using the `tidb-drainer` Helm chart for a TiDB cluster, take the following steps:

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

```
helm repo update
```

```
helm search repo 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: v5.0.6
baseImage:pingcap/tidb-binlog
storageClassName: local-storage
storage: 10Gi
initialCommitTs: "-1"
config: |
  detect-interval = 10
  [syncer]
  worker-count = 16
  txn-batch = 20
  disable-dispatch = false
  ignore-schemas = "INFORMATION_SCHEMA,PERFORMANCE_SCHEMA,mysql"
  safe-mode = false
  db-type = "tidb"
  [syncer.to]
  host = "downstream-tidb"
  user = "root"
  password = ""
  port = 4000
```

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

The `initialCommitTs` is the starting commit timestamp of data replication when Drainer has no checkpoint. The value must be set as a string type, such as `"424364429251444742"`.

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

4. Deploy Drainer:

```
helm install ${release_name} pingcap/tidb-drainer --namespace=${
    ↪ namespace} --version=${chart_version} -f values.yaml
```

If the server does not have an external network, refer to deploy the TiDB cluster to download the required Docker image on the machine with an external network and upload it to the server.

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

4.5.4 Enable TLS

4.5.4.1 Enable TLS between TiDB components

If you want to enable TLS for the TiDB cluster and TiDB Binlog, refer to Enable TLS between Components.

After you have created a secret and started a TiDB cluster with Pump, edit the `values.yaml` file to set the `tlsCluster.enabled` value to `true`, and configure the corresponding `certAllowedCN`:

```
...
tlsCluster:
  enabled: true
  # certAllowedCN:
  #  - TiDB
...
```

4.5.4.2 Enable TLS between Drainer and the downstream database

If you set the downstream database of `tidb-drainer` to `mysql/tidb`, and if you want to enable TLS between Drainer and the downstream database, take the following steps.

1. Create a secret that contains the TLS information of the downstream database.

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

`tidb-drainer` saves the checkpoint in the downstream database by default, so you only need to configure `tlsSyncer.tlsClientSecretName` and the corresponding `cerAllowedCN`:

```
tlsSyncer:
  tlsClientSecretName: ${downstream_database_secret_name}
  # certAllowedCN:
  #  - TiDB
```

2. To save the checkpoint of `tidb-drainer` to **other databases that have enabled TLS**, create a secret that contains the TLS information of the checkpoint database:

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

Edit the `values.yaml` file to set the `tlsSyncer.checkpoint.tlsClientSecretName` ↪ value to `${checkpoint_tidb_client_secret}`, and configure the corresponding `certAllowedCN`:

```
...
tlsSyncer: {}
  tlsClientSecretName: ${downstream_database_secret_name}
  # certAllowedCN:
  #  - TiDB
  checkpoint:
    tlsClientSecretName: ${checkpoint_tidb_client_secret}
    # certAllowedCN:
    #  - TiDB
...
```

4.5.5 Remove Pump/Drainer nodes

For details on how to maintain the node state of the TiDB Binlog cluster, refer to Starting and exiting a Pump or Drainer process.

If you want to remove the TiDB Binlog component completely, it is recommended that you first remove Pump nodes and then remove Drainer nodes.

If TLS is enabled for the TiDB Binlog component to be removed, write the following content into `binlog.yaml` and execute `kubectl apply -f binlog.yaml` to start a Pod that is mounted with the TLS file and the `binlogctl` tool.

```
apiVersion: v1
kind: Pod
metadata:
  name: binlogctl
spec:
  containers:
  - name: binlogctl
    image: pingcap/tidb-binlog:${tidb_version}
    command: ['/bin/sh']
    stdin: true
    stdinOnce: true
    tty: true
    volumeMounts:
      - name: binlog-tls
        mountPath: /etc/binlog-tls
  volumes:
    - name: binlog-tls
      secret:
        secretName: ${cluster_name}-cluster-client-secret
```

4.5.5.1 Scale in Pump

To scale in Pump, you need to take a single Pump node offline, and execute `kubectl ↪ edit tc ${cluster_name} -n ${namespace}` to reduce the value of `replicas` of Pump by 1. Repeat the operations on each node.

The steps are as follows:

1. Take the Pump node offline.

   Assume there are three Pump nodes in the cluster. You need to take the third node offline and replace `${ordinal_id}` with 2. (`${tidb_version}` is the current TiDB version.)

   - If TLS is not enabled for Pump, create a Pod to take Pump offline:

     ```
     kubectl run offline-pump-${ordinal_id} --image=pingcap/tidb-binlog:
         ↪ ${tidb_version} --namespace=${namespace} --restart=OnFailure
         ↪  -- /binlogctl -pd-urls=http://${cluster_name}-pd:2379 -cmd
         ↪ offline-pump -node-id ${cluster_name}-pump-${ordinal_id
         ↪ }:8250
     ```

   - If TLS is enabled for Pump, use the previously started Pod to take Pump offline:

```
kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls "
    ↪ https://${cluster_name}-pd:2379" -cmd offline-pump -node-id
    ↪ ${cluster_name}-pump-${ordinal_id}:8250 -ssl-ca "/etc/binlog
    ↪ -tls/ca.crt" -ssl-cert "/etc/binlog-tls/tls.crt" -ssl-key "/
    ↪ etc/binlog-tls/tls.key"
```

View the log of Pump by executing the following command:

```
kubectl logs -f -n ${namespace} ${release_name}-pump-${ordinal_id}
```

If `pump offline, please delete my pod` is output, this node is successfully taken offline.

2. Delete the corresponding Pump Pod:

   Execute `kubectl edit tc ${cluster_name} -n ${namespace}` to change `spec.pump.replicas` to 2, and wait until the Pump Pod is taken offline and deleted automatically.

3. (Optional) Force Pump to go offline:

   If the offline operation fails, the Pump Pod will not output `pump offline, please delete my pod`. At this time, you can force Pump to go offline, that is, taking Step 2 to reduce the value of `spec.pump.replicas` and mark Pump as `offline` after the Pump Pod is deleted completely.

   - If TLS is not enabled for Pump, mark Pump as `offline`:

     ```
     kubectl run update-pump-${ordinal_id} --image=pingcap/tidb-binlog:$
         ↪ {tidb_version} --namespace=${namespace} --restart=OnFailure
         ↪ -- /binlogctl -pd-urls=http://${cluster_name}-pd:2379 -cmd
         ↪ update-pump -node-id ${cluster_name}-pump-${ordinal_id}:8250
         ↪  --state offline
     ```

   - If TLS is enabled for Pump, mark Pump as `offline` using the previously started Pod:

     ```
     kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls=https
         ↪ ://${cluster_name}-pd:2379 -cmd update-pump -node-id ${
         ↪ cluster_name}-pump-${ordinal_id}:8250 --state offline -ssl-
         ↪ ca "/etc/binlog-tls/ca.crt" -ssl-cert "/etc/binlog-tls/tls.
         ↪ crt" -ssl-key "/etc/binlog-tls/tls.key"
     ```

4.5.5.2 Remove Pump nodes completely

Note:

- Before performing the following steps, you need to have at least one Pump node in the cluster. If you have scaled in Pump nodes to `0`, you need to scale out Pump at least to `1` node before you perform the removing operation in this section.
- To scale out the Pump to `1`, execute `kubectl edit tc ${tidb-cluster ↪ } -n ${namespace}` and modify the `spec.pump.replicas` to `1`.

1. Before removing Pump nodes, execute `kubectl edit tc ${cluster_name} -n ${ ↪ namespace}` and set `spec.tidb.binlogEnabled` to `false`. After the TiDB Pods are rolling updated, you can remove the Pump nodes.

   If you directly remove Pump nodes, it might cause TiDB failure because TiDB has no Pump nodes to write into.

2. Refer to Scale in Pump to scale in Pump to `0`.

3. Execute `kubectl edit tc ${cluster_name} -n ${namespace}` and delete all configuration items of `spec.pump`.

4. Execute `kubectl delete sts ${cluster_name}-pump -n ${namespace}` to delete the StatefulSet resources of Pump.

5. View PVCs used by the Pump cluster by executing `kubectl get pvc -n ${ ↪ namespace} -l app.kubernetes.io/component=pump`. Then delete all the PVC resources of Pump by executing `kubectl delete pvc -l app.kubernetes.io/ ↪ component=pump -n ${namespace}`.

4.5.5.3 Remove Drainer nodes

1. Take Drainer nodes offline:

   In the following commands, `${drainer_node_id}` is the node ID of the Drainer node to be taken offline. If you have configured `drainerName` in `values.yaml` of Helm, the value of `${drainer_node_id}` is `${drainer_name}-0`; otherwise, the value of `${ ↪ drainer_node_id}` is `${cluster_name}-${release_name}-drainer-0`.

   - If TLS is not enabled for Drainer, create a Pod to take Drainer offline:

```
kubectl run offline-drainer-0 --image=pingcap/tidb-binlog:${
    ↪ tidb_version} --namespace=${namespace} --restart=OnFailure
    ↪ -- /binlogctl -pd-urls=http://${cluster_name}-pd:2379 -cmd
    ↪ offline-drainer -node-id ${drainer_node_id}:8249
```

- If TLS is enabled for Drainer, use the previously started Pod to take Drainer offline:

```
kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls "
    ↪ https://${cluster_name}-pd:2379" -cmd offline-drainer -node-
    ↪ id ${drainer_node_id}:8249 -ssl-ca "/etc/binlog-tls/ca.crt"
    ↪ -ssl-cert "/etc/binlog-tls/tls.crt" -ssl-key "/etc/binlog-
    ↪ tls/tls.key"
```

View the log of Drainer by executing the following command:

```
kubectl logs -f -n ${namespace} ${drainer_node_id}
```

If `drainer offline, please delete my pod` is output, this node is successfully taken offline.

2. Delete the corresponding Drainer Pod:

   Execute `helm uninstall ${release_name} -n ${namespace}` to delete the Drainer Pod.

   If you no longer need Drainer, execute `kubectl delete pvc data-${drainer_node_id ↪ } -n ${namespace}` to delete the PVC resources of Drainer.

3. (Optional) Force Drainer to go offline:

   If the offline operation fails, the Drainer Pod will not output `drainer offline, ↪ please delete my pod`. At this time, you can force Drainer to go offline, that is, taking Step 2 to delete the Drainer Pod and mark Drainer as `offline`.

   - If TLS is not enabled for Drainer, mark Drainer as `offline`:

```
kubectl run update-drainer-${ordinal_id} --image=pingcap/tidb-
    ↪ binlog:${tidb_version} --namespace=${namespace} --restart=
    ↪ OnFailure -- /binlogctl -pd-urls=http://${cluster_name}-pd
    ↪ :2379 -cmd update-drainer -node-id ${drainer_node_id}:8249
    ↪ --state offline
```

   - If TLS is enabled for Drainer, use the previously started Pod to take Drainer offline:

```
kubectl exec binlogctl -n ${namespace} -- /binlogctl -pd-urls=https
    ↪ ://${cluster_name}-pd:2379 -cmd update-drainer -node-id ${
    ↪ drainer_node_id}:8249 --state offline -ssl-ca "/etc/binlog-
    ↪ tls/ca.crt" -ssl-cert "/etc/binlog-tls/tls.crt" -ssl-key "/
    ↪ etc/binlog-tls/tls.key"
```

4.6 Deploy Multiple Sets of TiDB Operator

This document describes how to deploy multiple sets of TiDB Operator to manage different TiDB clusters.

Note:

- Currently, you can only deploy multiple sets of `tidb-controller-manager` and `tidb-scheduler`. Deploying multiple sets of Advanced-StatefulSet controller and `tidb-admission-webhook` is not supported.
- If you have deployed multiple sets of TiDB Operator and only some of them enable Advanced StatefulSet, the same TidbCluster Custom Resource (CR) cannot be switched among these TiDB Operator.
- This feature is supported since v1.1.10.

4.6.1 Related parameters

To support deploying multiple sets of TiDB Operator, the following parameters are added to the `values.yaml` file in the `tidb-operator` chart:

- `appendReleaseSuffix`

  If this parameter is set to `true`, when you deploy TiDB Operator, the Helm chart automatically adds a suffix (`-{{ .Release.Name }}`) to the name of resources related to `tidb-controller-manager` and `tidb-scheduler`.

  For example, if you execute `helm install canary pingcap/tidb-operator ...`, the name of the `tidb-controller-manager` deployment is `tidb-controller-manager-canary`.

  If you need to deploy multiple sets of TiDB Operator, set this parameter to `true`.

  Default value: `false`.

- `controllerManager.create`

  Controls whether to create `tidb-controller-manager`.

  Default value: `true`.

- `controllerManager.selector`

  Sets the `-selector` parameter for `tidb-controller-manager`. The parameter is used to filter the CRs controlled by `tidb-controller-manager` according to the CR labels. If multiple selectors exist, the selectors are in `and` relationship.

  Default value: `[]` (`tidb-controller-manager` controls all CRs).

  Example:

```
selector:
- canary-release=v1
- k1==v1
- k2!=v2
```

- `scheduler.create`

  Controls whether to create `tidb-scheduler`.

  Default value: `true`.

4.6.2 Deploy

1. Deploy the first TiDB Operator.

   Refer to Deploy TiDB Operator to deploy the first TiDB Operator. Add the following configuration in the `values.yaml`:

   ```
   controllerManager:
     selector:
     - user=dev
   ```

2. Deploy the TiDB cluster.

   1. Refer to Configure the TiDB Cluster to configure the TidbCluster CR, and configure `labels` to match the `selector` set in the last step. For example:

      ```
      apiVersion: pingcap.com/v1alpha1
      kind: TidbCluster
      metadata:
        name: basic1
        labels:
          user: dev
      spec:
        ...
      ```

      If `labels` is not set when you deploy the TiDB cluster, you can configure `labels` by running the following command:

      ```
      kubectl -n ${namespace} label tidbcluster ${cluster_name} user=dev
      ```

   2. Refer to Deploy TiDB in General Kubernetes to deploy the TiDB cluster. Confirm that each component in the cluster is started normally.

3. Deploy the second TiDB Operator.

   Refer to Deploy TiDB Operator to deploy the second TiDB Operator without `tidb-scheduler`. Add the following configuration in the `values.yaml` file, and deploy

the second TiDB Operator (without `tidb-scheduler`) in **a different namespace** (such as `tidb-admin-qa`) with a **different Helm Release Name** (such as `helm install tidb-operator-qa ...`):

```
controllerManager:
  selector:
  - user=qa
appendReleaseSuffix: true
scheduler:
  create: false
advancedStatefulset:
  create: false
admissionWebhook:
  create: false
```

Note:

- It is recommended to deploy the new TiDB Operator in a separate namespace.
- Set `appendReleaseSuffix` to `true`.
- If you configure `scheduler.create: true`, a `tidb-scheduler` named `{{ .scheduler.schedulerName }}-{{.Release.Name}}` is created. To use this `tidb-scheduler`, you need to configure `spec.schedulerName` in the `TidbCluster` CR to the name of this scheduler.
- You need to set `advancedStatefulset.create: false` and `admissionWebhook.create: false`, because deploying multiple sets of AdvancedStatefulSet controller and `tidb-admission-webhook` is not supported.

4. Deploy the TiDB cluster.

    1. Refer to Configure the TiDB Cluster to configure the TidbCluster CR, and configure `labels` to match the `selector` set in the last step. For example:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic2
  labels:
    user: qa
spec:
  ...
```

If `labels` is not set when you deploy the TiDB cluster, you can configure `labels` by running the following command:

```
kubectl -n ${namespace} label tidbcluster ${cluster_name} user=qa
```

2. Refer to Deploy TiDB in General Kubernetes to deploy the TiDB cluster. Confirm that each component in the cluster is started normally.

5. View the logs of the two sets of TiDB Operator, and confirm that each TiDB Operator manages the TiDB cluster that matches the corresponding selectors.

   For example:

   View the log of `tidb-controller-manager` of the first TiDB Operator:

```
kubectl -n tidb-admin logs tidb-controller-manager-55b887bdc9-lzdwv
```

   Output

   View the log of `tidb-controller-manager` of the second TiDB Operator:

```
kubectl -n tidb-admin-qa logs tidb-controller-manager-qa-5dfcd7f9-vll4c
```

   Output

   By comparing the logs of the two sets of TiDB Operator, you can confirm that the first TiDB Operator only manages the `tidb-cluster-1/basic1` cluster, and the second TiDB Operator only manages the `tidb-cluster-2/basic2` cluster.

4.7 Deploy Monitoring

4.7.1 Deploy Monitoring and Alerts for a TiDB Cluster

This document describes how to monitor a TiDB cluster deployed using TiDB Operator and configure alerts for the cluster.

4.7.1.1 Monitor the TiDB cluster

You can monitor the TiDB cluster with Prometheus and Grafana. When you create a new TiDB cluster using TiDB Operator, you can deploy a separate monitoring system for the TiDB cluster. The monitoring system must run in the same namespace as the TiDB cluster, and includes two components: Prometheus and Grafana.

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

In TiDB Operator v1.1 or later versions, you can monitor a TiDB cluster on a Kubernetes cluster by using a simple Custom Resource (CR) file called `TidbMonitor`.

Note:

- One `TidbMonitor` can only monitor one `TidbCluster`.
- `spec.clusters[0].name` should be set to the `TidbCluster` name of the corresponding TiDB cluster.

4.7.1.1.1 Persist monitoring data

The monitoring data is not persisted by default. To persist the monitoring data, you can set `spec.persistent` to `true` in `TidbMonitor`. When you enable this option, you need to set `spec.storageClassName` to an existing storage in the current cluster. This storage must support persisting data; otherwise, there is a risk of data loss.

A configuration example is as follows:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: basic
spec:
  clusters:
    - name: basic
  persistent: true
  storageClassName: ${storageClassName}
  storage: 5G
  prometheus:
    baseImage: prom/prometheus
    version: v2.18.1
    service:
      type: NodePort
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: NodePort
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

To verify the PVC status, run the following command:

```
kubectl get pvc -l app.kubernetes.io/instance=basic,app.kubernetes.io/
    ↪ component=monitor -n ${namespace}
```

```
NAME          STATUS  VOLUME                                     CAPACITY  ACCESS
    ↪  MODES STORAGECLASS AGE
basic-monitor Bound   pvc-6db79253-cc9e-4730-bbba-ba987c29db6f 5G     RWO
    ↪           standard     51s
```

4.7.1.1.2 Customize the Prometheus configuration

You can customize the Prometheus configuration by using a customized configuration file or by adding extra options to the command.

Use a customized configuration file

1. Create a ConfigMap for your customized configuration, and set the key name of `data` to `prometheus-config`.
2. Set `spec.prometheus.config.configMapRef.name` and `spec.prometheus.config.` ↪ `configMapRef.namespace` to the name and namespace of the customized ConfigMap respectively.

For the complete configuration, refer to the tidb-operator example.

Add extra options to the command

To add extra options to the command that starts Prometheus, configure `spec.` ↪ `prometheus.config.commandOptions`.

For the complete configuration, refer to the tidb-operator example.

Note:

The following options are automatically configured by the TidbMonitor controller and cannot be specified again via `commandOptions`.

- `config.file`
- `log.level`
- `web.enable-admin-api`
- `web.enable-lifecycle`
- `storage.tsdb.path`
- `storage.tsdb.retention`
- `storage.tsdb.max-block-duration`
- `storage.tsdb.min-block-duration`

4.7.1.1.3 Access the Grafana monitoring dashboard

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

```
kubectl port-forward -n ${namespace} svc/${cluster_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`.

You can also set `spec.grafana.service.type` to `NodePort` or `LoadBalancer`, and then view the monitoring dashboard through `NodePort` or `LoadBalancer`.

If there is no need to use Grafana, you can delete the part of `spec.grafana` in `TidbMonitor` during deployment. In this case, you need to use other existing or newly deployed data visualization tools to directly access the monitoring data.

4.7.1.1.4 Access the Prometheus monitoring data

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

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-prometheus
    ↪ 9090:9090 &>/tmp/portforward-prometheus.log &
```

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

You can also set `spec.prometheus.service.type` to `NodePort` or `LoadBalancer`, and then view the monitoring data through `NodePort` or `LoadBalancer`.

4.7.1.1.5 Set kube-prometheus and AlertManager

In some cases, TidbMonitor needs to obtain the monitoring metrics on Kubernetes. To obtain the kube-prometheus metrics, configure `TidbMonitor.Spec.kubePrometheusURL`.

Similarly, you can configure TidbMonitor to push the monitoring alert to AlertManager.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: basic
spec:
  clusters:
    - name: basic
  kubePrometheusURL: http://prometheus-k8s.monitoring:9090
  alertmanagerURL: alertmanager-main.monitoring:9093
  prometheus:
    baseImage: prom/prometheus
    version: v2.18.1
```

```
    service:
      type: NodePort
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: NodePort
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

4.7.1.2 Enable Ingress

This section introduces how to enable Ingress for TidbMonitor. Ingress is an API object that exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.

4.7.1.2.1 Prerequisites

Before using Ingress, you need to install the Ingress controller. Simply creating the Ingress resource does not take effect.

You need to deploy the NGINX Ingress controller, or choose from various Ingress controllers.

For more information, see Ingress Prerequisites.

4.7.1.2.2 Access TidbMonitor using Ingress

Currently, TidbMonitor provides a method to expose the Prometheus/Grafana service through Ingress. For details about Ingress, see Ingress official documentation.

The following example shows how to enable Prometheus and Grafana in TidbMonitor:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ingress-demo
spec:
  clusters:
    - name: demo
  persistent: false
  prometheus:
    baseImage: prom/prometheus
```

```
    version: v2.18.1
    ingress:
      hosts:
      - example.com
      annotations:
        foo: "bar"
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: ClusterIP
    ingress:
      hosts:
        - example.com
      annotations:
        foo: "bar"
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

To modify the setting of Ingress Annotations, configure `spec.prometheus.ingress.annotations` and `spec.grafana.ingress.annotations`. If you use the default Nginx Ingress, see Nginx Ingress Controller Annotation for details.

The TidbMonitor Ingress setting also supports TLS. The following example shows how to configure TLS for Ingress. See Ingress TLS for details.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ingress-demo
spec:
  clusters:
    - name: demo
  persistent: false
  prometheus:
    baseImage: prom/prometheus
    version: v2.18.1
    ingress:
      hosts:
      - example.com
      tls:
```

```
    - hosts:
      - example.com
      secretName: testsecret-tls
  grafana:
    baseImage: grafana/grafana
    version: 6.1.6
    service:
      type: ClusterIP
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.0.6
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
```

TLS Secret must include the `tls.crt` and `tls.key` keys, which include the certificate and private key used for TLS. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: ${namespace}
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

In a public cloud-deployed Kubernetes cluster, you can usually configure Loadbalancer to access Ingress through a domain name. If you cannot configure the Loadbalancer service (for example, when you use NodePort as the service type of Ingress), you can access the service in a way equivalent to the following command:

```
curl -H "Host: example.com" ${node_ip}:${NodePort}
```

4.7.1.3 Configure alert

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.

The custom configuration of alert rules is supported. You can modify the alert rules by taking the following steps:

1. When deploying the monitoring system for the TiDB cluster, set `spec.reloader.service.type` to `NodePort` or `LoadBalancer`.

2. Access the `reloader` service through `NodePort` or `LoadBalancer`. Click the `Files` ↪ button above to select the alert rule file to be modified, and make the custom configuration. Click `Save` after the modification.

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 set the value of `spec.alertmanagerURL` to the address of `AlertManager`, which will be used by Prometheus. For details, refer to Set kube-prometheus and AlertManager.
- If no AlertManager service is available, or if you want to deploy a separate AlertManager service, refer to the Prometheus official document.

4.7.2 Access TiDB Dashboard

> **Warning:**
>
> The TiDB Dashboard is available in the `/dashboard` path of the PD. Other paths outside of this may not have access control.

TiDB Dashboard is a visualized tool introduced starting from TiDB 4.0 and is used to monitor and diagnose TiDB clusters. For details, see TiDB Dashboard.

This document describes how to access TiDB Dashboard in Kubernetes.

> **Note:**
>
> TiDB Operator starts a Discovery service for each TiDB cluster. The Discovery service can return the corresponding startup parameters for each PD Pod to support the startup of the PD cluster. The Discovery service can also send proxy requests to the TiDB Dashboard. In this document, TiDB Dashboard is accessed using the Discovery service.

4.7.2.1 Prerequisites

To access TiDB Dashboard smoothly in Kubernetes, you need to use TiDB Operator v1.1.1 (or later versions) and the TiDB cluster (v4.0.1 or later versions).

You need to configure the `TidbCluster` object file as follows to enable quick access to TiDB Dashboard:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  pd:
    enableDashboardInternalProxy: true
```

4.7.2.2 Access TiDB Dashboard by port forward

> **Warning:**
>
> This guide shows how to quickly access TiDB Dashboard. Do **NOT** use this method in the production environment. For production environments, refer to Access TiDB Dashboard by Ingress.

TiDB Dashboard is built in the PD component in TiDB 4.0 and later versions. You can refer to the following example to quickly deploy a v4.0.4 TiDB cluster in Kubernetes.

1. Deploy the following `.yaml` file into the Kubernetes cluster by running the `kubectl apply -f` command:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    enableDashboardInternalProxy: true
    baseImage: pingcap/pd
    replicas: 1
    requests:
```

```
    storage: "1Gi"
  config: {}
tikv:
  baseImage: pingcap/tikv
  replicas: 1
  requests:
    storage: "1Gi"
  config: {}
tidb:
  baseImage: pingcap/tidb
  replicas: 1
  service:
    type: ClusterIP
  config: {}
```

2. After the cluster is created, expose TiDB Dashboard to the local machine by running the following command:

```
kubectl port-forward svc/basic-discovery -n ${namespace} 10262:10262
```

3. Visit http://localhost:10262/dashboard in your browser to access TiDB Dashboard.

> **Note:**
>
> By default, `port-forward` binds to the IP address 127.0.0.1. If you need to use another IP address to access the machine running the `port-forward` command, you can add the `-address` option and specify the IP address to be bound.

4.7.2.3 Access TiDB Dashboard by Ingress

In important production environments, it is recommended to expose the TiDB Dashboard service using Ingress.

4.7.2.3.1 Prerequisites

Before using Ingress, install the Ingress controller in your Kubernetes cluster. Otherwise, simply creating Ingress resources does not take effect.

To deploy the Ingress controller, refer to ingress-nginx. You can also choose from various Ingress controllers.

4.7.2.3.2 Use Ingress

You can expose the TiDB Dashboard service outside the Kubernetes cluster by using Ingress. In this way, the service can be accessed outside Kubernetes via `http/https`. For more details, see Ingress.

The following is an `.yaml` example of accessing TiDB Dashboard using Ingress:

1. Deploy the following `.yaml` file to the Kubernetes cluster by running the `kubectl ↪ apply -f` command:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: access-dashboard
  namespace: ${namespace}
spec:
  rules:
    - host: ${host}
      http:
        paths:
          - backend:
              serviceName: ${cluster_name}-discovery
              servicePort: 10262
            path: /dashboard
```

2. After Ingress is deployed, you can access TiDB Dashboard via http://$%7Bhost%7D/dashboard outside the Kubernetes cluster.

4.7.2.4 Enable Ingress TLS

Ingress supports TLS. See Ingress TLS. The following example shows how to use Ingress TLS:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: access-dashboard
  namespace: ${namespace}
spec:
  tls:
  - hosts:
    - ${host}
    secretName: testsecret-tls
  rules:
    - host: ${host}
      http:
```

```
      paths:
        - backend:
            serviceName: ${cluster_name}-discovery
            servicePort: 10262
          path: /dashboard
```

In the above file, `testsecret-tls` contains `tls.crt` and `tls.key` needed for `example.com`.

This is an example of `testsecret-tls`:

```
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

After Ingress is deployed, visit https://%7Bhost%7D/dashboard to access TiDB Dashboard.

4.7.2.4.1 Use NodePort Service

Because `ingress` can only be accessed with a domain name, it might be difficult to use `ingress` in some scenarios. In this case, to access and use TiDB Dashboard, you can add a `Service` of `NodePort` type.

The following is an `.yaml` example using the `Service` of `NodePort` type to access the TiDB Dashboard. To deploy the following `.yaml` file into the Kubernetes cluster, you can run the `kubectl apply -f` command:

```
apiVersion: v1
kind: Service
metadata:
  name: access-dashboard
  namespace: ${namespace}
spec:
  ports:
  - name: dashboard
    port: 10262
    protocol: TCP
    targetPort: 10262
  type: NodePort
  selector:
```

```
    app.kubernetes.io/component: discovery
    app.kubernetes.io/instance: ${cluster_name}
    app.kubernetes.io/name: tidb-cluster
```

After deploying the `Service`, you can access TiDB Dashboard via https://%7BnodeIP%7D:%7BnodePort%7D/dashboard. By default, `nodePort` is randomly assigned by Kubernetes. You can also specify an available port in the `.yaml` file.

Note that if there is more than one PD `Pod` in the cluster, you need to set `spec.pd.enableDashboardInternalProxy: true` in the `TidbCluster` CR to ensure normal access to TiDB Dashboard.

4.7.2.5 Update the TiDB cluster

To enable quick access to TiDB Dashboard by updating an existing TiDB cluster, update the following two configurations:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  configUpdateStrategy: RollingUpdate
  pd:
    enableDashboardInternalProxy: true
```

4.7.2.6 Unsupported TiDB Dashboard features

Due to the special environment of Kubernetes, some features of TiDB Dashboard are not supported in TiDB Operator, including:

- In **Overview** -> **Monitor & Alert** -> **View Metrics**, the link does not direct to the Grafana monitoring dashboard. If you need to access Grafana, refer to Access the Grafana monitoring dashboard.
- The log search feature is unavailable. If you need to view the log of a component, execute `kubectl logs ${pod_name} -n {namespace}`. You can also view logs using the log service of the Kubernetes cluster.
- In **Cluster Info** -> **Hosts**, the **Disk Usage** cannot display correctly. You can view the disk usage of each component by viewing the component dashboards in the Tidb-Monitor dashboard. You can also view the disk usage of Kubernetes nodes by deploying a Kubernetes host monitoring system.

5 Secure

5.1 Enable TLS for the MySQL Client

This document describes how to enable TLS for MySQL client of the TiDB cluster on Kubernetes. Starting from TiDB Operator v1.1, TLS for the MySQL client of the TiDB cluster on Kubernetes is supported.

To enable TLS for the MySQL client, perform the following steps:

1. Issue two sets of certificates: a set of server-side certificates for TiDB server, and a set of client-side certificates for MySQL client. Create two Secret objects, `${cluster_name}-tidb-server-secret` and `${cluster_name}-tidb-client-secret`, respectively including these two sets of certificates.

> **Note:**
> The Secret objects you created must follow the above naming convention. Otherwise, the deployment of the TiDB cluster will fail.

   Certificates can be issued in multiple methods. This document describes two methods. You can choose either of them to issue certificates for the TiDB cluster:

   - Using the `cfssl` system
   - Using the `cert-manager` system

2. Deploy the cluster, and set `.spec.tidb.tlsClient.enabled` to `true`.
3. Configure the MySQL client to use an encrypted connection.

If you need to renew the existing TLS certificate, refer to Renew and Replace the TLS Certificate.

5.1.1 Issue two sets of certificates for the TiDB cluster

This section describes how to issue certificates for the TiDB cluster using two methods: `cfssl` and `cert-manager`.

5.1.1.1 Using `cfssl`

1. Download `cfssl` and initialize the certificate issuer:

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

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

2. Configure the client auth (CA) option in `ca-config.json`:

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

3. Change the certificate signing request (CSR) of `ca-csr.json`:

```
{
    "CN": "TiDB Server",
    "CA": {
```

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

4. Generate CA by the configured option:

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

5. Generate the server-side certificate:

   First, create the default `server.json` file:

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

   Then, edit this file to change the `CN`, `hosts` attributes:

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

`${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

Finally, generate the server-side certificate:

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

6. Generate the client-side certificate:

   First, create the default `client.json` file:

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

   Then, edit this file to change the `CN`, `hosts` attributes. You can leave the `hosts` empty:

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

   Finally, generate the client-side certificate:

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

7. Create the Kubernetes Secret object.

   If you have already generated two sets of certificates as described in the above steps, create the Secret object for the TiDB cluster by the following command:

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

   You have created two Secret objects for the server-side and client-side certificates:

   - The TiDB server loads one Secret object when it starts
   - The MySQL client uses another Secret object when it connects to the TiDB cluster

You can generate multiple sets of client-side certificates. At least one set of client-side certificates is needed for the internal components of TiDB Operator to access the TiDB server. Currently, `TidbInitializer` accesses the TiDB server to set the password or perform initialization.

5.1.1.2 Using `cert-manager`

1. Install `cert-manager`.

   Refer to cert-manager installation in Kubernetes.

2. Create an Issuer to issue certificates for the TiDB cluster.

   To configure `cert-manager`, create the Issuer resources.

   First, create a directory which saves the files that `cert-manager` needs to create certificates:

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

   Then, create a `tidb-server-issuer.yaml` file with the following content:

```
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-ca
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-ca-secret
  commonName: "TiDB CA"
  isCA: true
  duration: 87600h # 10yrs
  renewBefore: 720h # 30d
  issuerRef:
    name: ${cluster_name}-selfsigned-ca-issuer
    kind: Issuer
---
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-tidb-issuer
  namespace: ${namespace}
spec:
  ca:
```

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

This `.yaml` file creates three objects:

- An Issuer object of SelfSigned class, used to generate the CA certificate needed by Issuer of CA class
- A Certificate object, whose `isCa` is set to `true`
- An Issuer, used to issue TLS certificates for the TiDB server

Finally, execute the following command to create an Issuer:

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

3. Generate the server-side certificate.

   In `cert-manager`, the Certificate resource represents the certificate interface. This certificate is issued and updated by the Issuer created in Step 2.

   First, create a `tidb-server-cert.yaml` file with the following content:

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-server-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-server-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "TiDB Server"
  usages:
    - server auth
  dnsNames:
    - "${cluster_name}-tidb"
    - "${cluster_name}-tidb.${namespace}"
    - "${cluster_name}-tidb.${namespace}.svc"
    - "*.${cluster_name}-tidb"
    - "*.${cluster_name}-tidb.${namespace}"
    - "*.${cluster_name}-tidb.${namespace}.svc"
    - "*.${cluster_name}-tidb-peer"
    - "*.${cluster_name}-tidb-peer.${namespace}"
    - "*.${cluster_name}-tidb-peer.${namespace}.svc"
  ipAddresses:
    - 127.0.0.1
    - ::1
```

```
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

`${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${cluster_name}-tidb-server-secret`
- Add `server auth` in `usages`
- Add the following 6 DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  - `${cluster_name}-tidb`
  - `${cluster_name}-tidb.${namespace}`
  - `${cluster_name}-tidb.${namespace}.svc`
  - `*.${cluster_name}-tidb`
  - `*.${cluster_name}-tidb.${namespace}`
  - `*.${cluster_name}-tidb.${namespace}.svc`
  - `*.${cluster_name}-tidb-peer`
  - `*.${cluster_name}-tidb-peer.${namespace}`
  - `*.${cluster_name}-tidb-peer.${namespace}.svc`
- Add the following 2 IPs in `ipAddresses`. You can also add other IPs according to your needs:
  - `127.0.0.1`
  - `::1`
- Add the Issuer created above in the `issuerRef`
- For other attributes, refer to cert-manager API.

Execute the following command to generate the certificate:

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

After the object is created, cert-manager generates a `${cluster_name}-tidb-server-secret` Secret object to be used by the TiDB server.

4. Generate the client-side certificate:

   Create a `tidb-client-cert.yaml` file with the following content:

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-client-secret
  duration: 8760h # 365d
```

```
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "TiDB Client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

`${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${cluster_name}-tidb-client-secret`
- Add `client auth` in `usages`
- `dnsNames` and `ipAddresses` are not required
- Add the Issuer created above in the `issuerRef`
- For other attributes, refer to cert-manager API

Execute the following command to generate the certificate:

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

After the object is created, cert-manager generates a `${cluster_name}-tidb-client-secret` Secret object to be used by the TiDB client.

5. Create multiple sets of client-side certificates (optional).

   Four components in the TiDB Operator cluster need to request the TiDB server. When TLS is enabled, these components can use certificates to request the TiDB server, each with a separate certificate. The four components are listed as follows:

   - TidbInitializer
   - PD Dashboard
   - Backup
   - Restore

   If you need to restore data using TiDB Lightning, you need to generate a server-side certificate for the TiDB Lightning component.

   To create certificates for these components, take the following steps:

   1. Create a `tidb-components-client-cert.yaml` file with the following content:

      ```
      apiVersion: cert-manager.io/v1alpha2
      kind: Certificate
      metadata:
        name: ${cluster_name}-tidb-initializer-client-secret
      ```

```
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-initializer-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "TiDB Initializer client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-pd-dashboard-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-pd-dashboard-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "PD Dashboard client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-backup-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-backup-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
```

```
  commonName: "Backup client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-restore-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-restore-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "Restore client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

In the `.yaml` file above, `${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set the value of `spec.secretName` to `${cluster_name}-${component}-client-secret`.
- Add `client auth` in `usages`.
- `dnsNames` and `ipAddresses` are not required.
- Add the Issuer created above in the `issuerRef`.
- For other attributes, refer to cert-manager API.

To generate a client-side certificate for TiDB Lightning, use the following content and set `tlsCluster.tlsClientSecretName` to `${cluster_name}-lightning-client-secret` in TiDB Lightning's `values.yaml` file.

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-lightning-client-secret
  namespace: ${namespace}
spec:
```

```
  secretName: ${cluster_name}-lightning-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
    - PingCAP
  commonName: "Lightning client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

2. Create the certificate by running the following command:

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

3. After creating these objects, cert-manager will generate four secret objects for the four components.

> **Note:**
> TiDB server's TLS is compatible with the MySQL protocol. When the certificate content is changed, the administrator needs to manually execute the SQL statement `alter instance reload tls` to refresh the content.

5.1.2 Deploy the TiDB cluster

In this step, you create a TiDB cluster and perform the following operations:

- Enable TLS for the MySQL client
- Initialize the cluster (an `app` database is created for demonstration)
- Create a Backup object to back up the cluster
- Create a Restore object to restore the cluster
- Use separate client-side certificates for `TidbInitializer`, PD Dashboard, Backup, and Restore (specified by `tlsClientSecretName`)

1. Create three `.yaml` files:

    - `tidb-cluster.yaml` file:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Retain
  pd:
    baseImage: pingcap/pd
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
    tlsClientSecretName: ${cluster_name}-pd-dashboard-client-secret
  tikv:
    baseImage: pingcap/tikv
    replicas: 1
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 1
    service:
      type: ClusterIP
    config: {}
    tlsClient:
      enabled: true
---
apiVersion: pingcap.com/v1alpha1
kind: TidbInitializer
metadata:
  name: ${cluster_name}-init
  namespace: ${namespace}
spec:
  image: tnir/mysqlclient
  cluster:
    namespace: ${namespace}
    name: ${cluster_name}
  initSql: |-
    create database app;
  tlsClientSecretName: ${cluster_name}-tidb-initializer-client-
```

```
    ↪ secret
```

- backup.yaml:

```
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: ${cluster_name}-backup
  namespace: ${namespace}
spec:
  backupType: full
  br:
  cluster: ${cluster_name}
  clusterNamespace: ${namespace}
  sendCredToTikv: true
  from:
    host: ${host}
    secretName: ${tidb_secret}
    port: 4000
    user: root
    tlsClientSecretName: ${cluster_name}-backup-client-secret
  s3:
    provider: aws
    region: ${my_region}
    secretName: ${s3_secret}
    bucket: ${my_bucket}
    prefix: ${my_folder}
```

- restore.yaml:

```
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: ${cluster_name}-restore
  namespace: ${namespace}
spec:
  backupType: full
  br:
    cluster: ${cluster_name}
    clusterNamespace: ${namespace}
    sendCredToTikv: true
  to:
    host: ${host}
    secretName: ${tidb_secret}
    port: 4000
    user: root
```

```
  tlsClientSecretName: ${cluster_name}-restore-client-secret
s3:
  provider: aws
  region: ${my_region}
  secretName: ${s3_secret}
  bucket: ${my_bucket}
  prefix: ${my_folder}
```

In the above file, `${cluster_name}` is the name of the cluster, and `${namespace}` is the namespace in which the TiDB cluster is deployed. To enable TLS for the MySQL client, set `spec.tidb.tlsClient.enabled` to `true`.

2. Deploy the TiDB cluster:

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

3. Back up the cluster:

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

4. Restore the cluster:

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

5.1.3 Configure the MySQL client to use an encrypted connection

To connect the MySQL client with the TiDB cluster, use the client-side certificate created above and take the following methods. For details, refer to Configure the MySQL client to use encrypted connections.

Execute the following command to acquire the client-side certificate and connect to the TiDB server:

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

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

Note:

The default authentication plugin of MySQL 8.0 is updated from `mysql_native_password` to `caching_sha2_password`. Therefore, if you use MySQL client from MySQL 8.0 to access the TiDB service (TiDB version < v4.0.7), and if the user account has a password, you need to explicitly specify the `--default-auth=mysql_native_password` parameter.

Finally, to verify whether TLS is successfully enabled, refer to checking the current connection.

5.2 Enable TLS between TiDB Components

This document describes how to enable Transport Layer Security (TLS) between components of the TiDB cluster in Kubernetes, which is supported since TiDB Operator v1.1.

To enable TLS between TiDB components, perform the following steps:

1. Generate certificates for each component of the TiDB cluster to be created:

- A set of server-side certificates for the PD/TiKV/TiDB/Pump/Drainer/TiFlash/TiKV Importer/TiDB Lightning component, saved as the Kubernetes Secret objects: `${cluster_name}-${component_name}-cluster-secret`.
- A set of shared client-side certificates for the various clients of each component, saved as the Kubernetes Secret objects: `${cluster_name}-cluster-client-secret`.

Note:

The Secret objects you created must follow the above naming convention. Otherwise, the deployment of the TiDB components will fail.

2. Deploy the cluster, and set `.spec.tlsCluster.enabled` to `true`.
3. Configure `pd-ctl` and `tikv-ctl` to connect to the cluster.

Note:

- TiDB 4.0.5 (or later versions) and TiDB Operator 1.1.4 (or later versions) support enabling TLS for TiFlash.
- TiDB 4.0.3 (or later versions) and TiDB Operator 1.1.3 (or later versions) support enabling TLS for TiCDC.

Certificates can be issued in multiple methods. This document describes two methods. You can choose either of them to issue certificates for the TiDB cluster:

- Using the `cfssl` system
- Using the `cert-manager` system

If you need to renew the existing TLS certificate, refer to Renew and Replace the TLS Certificate.

5.2.1 Generate certificates for components of the TiDB cluster

This section describes how to issue certificates using two methods: `cfssl` and `cert-manager`.

5.2.1.1 Using `cfssl`

1. Download `cfssl` and initialize the certificate issuer:

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

mkdir -p cfssl
cd cfssl
```

2. Generate the `ca-config.json` configuration file:

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

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

3. Generate the `ca-csr.json` configuration file:

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

4. Generate CA by the configured option:

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

5. Generate the server-side certificates:

In this step, a set of server-side certificate is created for each component of the TiDB cluster.

- PD

  First, generate the default `pd-server.json` file:

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

  Then, edit this file to change the `CN` and `hosts` attributes:

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

  `${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

  Finally, generate the PD server-side certificate:

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

- TiKV

  First, generate the default `tikv-server.json` file:

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

  Then, edit this file to change the `CN` and `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${cluster_name}-tikv",
      "${cluster_name}-tikv.${namespace}",
```

```
      "${cluster_name}-tikv.${namespace}.svc",
      "${cluster_name}-tikv-peer",
      "${cluster_name}-tikv-peer.${namespace}",
      "${cluster_name}-tikv-peer.${namespace}.svc",
      "*.${cluster_name}-tikv-peer",
      "*.${cluster_name}-tikv-peer.${namespace}",
      "*.${cluster_name}-tikv-peer.${namespace}.svc"
    ],
...
```

`${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

Finally, generate the TiKV server-side certificate:

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

- TiDB

  First, create the default `tidb-server.json` file:

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

  Then, edit this file to change the `CN`, `hosts` attributes:

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

  `${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

  Finally, generate the TiDB server-side certificate:

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

- Pump

  First, create the default `pump-server.json` file:

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

  Then, edit this file to change the `CN`, `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "*.${cluster_name}-pump",
      "*.${cluster_name}-pump.${namespace}",
      "*.${cluster_name}-pump.${namespace}.svc"
    ],
...
```

  `${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

  Finally, generate the Pump server-side certificate:

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

- Drainer

  First, generate the default `drainer-server.json` file:

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

  Then, edit this file to change the `CN`, `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "<for hosts list, see the following instructions>"
    ],
...
```

  Drainer is deployed using Helm. The `hosts` field varies with different configuration of the `values.yaml` file.

  If you have set the `drainerName` attribute when deploying Drainer as follows:

```
...
# Changes the names of the statefulset and Pod.
# The default value is clusterName-ReleaseName-drainer.
# Does not change the name of an existing running Drainer, which is
   ↪  unsupported.
drainerName: my-drainer
...
```

Then you can set the `hosts` attribute as described below:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "*.${drainer_name}",
      "*.${drainer_name}.${namespace}",
      "*.${drainer_name}.${namespace}.svc"
    ],
...
```

If you have not set the `drainerName` attribute when deploying Drainer, configure the `hosts` attribute as follows:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "*.${cluster_name}-${release_name}-drainer",
      "*.${cluster_name}-${release_name}-drainer.${namespace}",
      "*.${cluster_name}-${release_name}-drainer.${namespace}.svc"
    ],
...
```

`${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. `${release_name}` is the `release name` you set when `helm install` is executed. `${drainer_name}` is `drainerName` in the `values.yaml` file. You can also add your customized `hosts`.

Finally, generate the Drainer server-side certificate:

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

- TiCDC

1. Generate the default `ticdc-server.json` file:

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

2. Edit this file to change the `CN`, `hosts` attributes:

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

`${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

3. Generate the TiCDC server-side certificate:

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

- TiFlash

1. Generate the default `tiflash-server.json` file:

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

2. Edit this file to change the `CN` and `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${cluster_name}-tiflash",
      "${cluster_name}-tiflash.${namespace}",
      "${cluster_name}-tiflash.${namespace}.svc",
      "${cluster_name}-tiflash-peer",
```

```
      "${cluster_name}-tiflash-peer.${namespace}",
      "${cluster_name}-tiflash-peer.${namespace}.svc",
      "*.${cluster_name}-tiflash-peer",
      "*.${cluster_name}-tiflash-peer.${namespace}",
      "*.${cluster_name}-tiflash-peer.${namespace}.svc"
    ],
...
```

`${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

3. Generate the TiFlash server-side certificate:

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

- TiKV Importer

  If you need to restore data using TiDB Lightning, you need to generate a server-side certificate for the TiKV Importer component.

  1. Generate the default `importer-server.json` file:

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

  2. Edit this file to change the `CN` and `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${cluster_name}-importer",
      "${cluster_name}-importer.${namespace}",
      "${cluster_name}-importer.${namespace}.svc"
      "${cluster_name}-importer.${namespace}.svc",
      "*.${cluster_name}-importer",
      "*.${cluster_name}-importer.${namespace}",
      "*.${cluster_name}-importer.${namespace}.svc"
    ],
...
```

  `${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

  3. Generate the TiKV Importer server-side certificate:

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

- TiDB Lightning

  If you need to restore data using TiDB Lightning, you need to generate a server-side certificate for the TiDB Lightning component.

  1. Generate the default `lightning-server.json` file:

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

  2. Edit this file to change the `CN` and `hosts` attributes:

     ```
     ...
         "CN": "TiDB",
         "hosts": [
           "127.0.0.1",
           "::1",
           "${cluster_name}-lightning",
           "${cluster_name}-lightning.${namespace}",
           "${cluster_name}-lightning.${namespace}.svc"
         ],
     ...
     ```

     `${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

  3. Generate the TiDB Lightning server-side certificate:

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

6. Generate the client-side certificate:

   First, create the default `client.json` file:

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

   Then, edit this file to change the `CN`, `hosts` attributes. You can leave the `hosts` empty:

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

   Finally, generate the client-side certificate:

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

7. Create the Kubernetes Secret object:

   If you have already generated a set of certificates for each component and a set of client-side certificate for each client as described in the above steps, create the Secret objects for the TiDB cluster by executing the following command:

   - The PD cluster certificate Secret:

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

   - The TiKV cluster certificate Secret:

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

   - The TiDB cluster certificate Secret:

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

   - The Pump cluster certificate Secret:

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

   - The Drainer cluster certificate Secret:

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

   - The TiCDC cluster certificate Secret:

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

- The TiFlash cluster certificate Secret:

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

- The TiKV Importer cluster certificate Secret:

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

- The TiDB Lightning cluster certificate Secret:

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

- The client certificate Secret:

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

You have created two Secret objects:

- One Secret object for each PD/TiKV/TiDB/Pump/Drainer server-side certificate to load when the server is started;
- One Secret object for their clients to connect.

5.2.1.2 Using `cert-manager`

1. Install `cert-manager`.

   Refer to cert-manager installation in Kubernetes for details.

2. Create an Issuer to issue certificates to the TiDB cluster.

   To configure `cert-manager`, create the Issuer resources.

   First, create a directory which saves the files that `cert-manager` needs to create certificates:

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

Then, create a `tidb-cluster-issuer.yaml` file with the following content:

```
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-ca
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-ca-secret
  commonName: "TiDB"
  isCA: true
  duration: 87600h # 10yrs
  renewBefore: 720h # 30d
  issuerRef:
    name: ${cluster_name}-selfsigned-ca-issuer
    kind: Issuer
---
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
  name: ${cluster_name}-tidb-issuer
  namespace: ${namespace}
spec:
  ca:
    secretName: ${cluster_name}-ca-secret
```

`${cluster_name}` is the name of the cluster. The above YAML file creates three objects:

- An Issuer object of the SelfSigned type, used to generate the CA certificate needed by Issuer of the CA type;
- A Certificate object, whose `isCa` is set to `true`.
- An Issuer, used to issue TLS certificates between TiDB components.

Finally, execute the following command to create an Issuer:

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

3. Generate the server-side certificate.

   In `cert-manager`, the Certificate resource represents the certificate interface. This certificate is issued and updated by the Issuer created in Step 2.

   According to Enable TLS Authentication, each component needs a server-side certificate, and all components need a shared client-side certificate for their clients.

   - PD

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-pd-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-pd-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-pd"
  - "${cluster_name}-pd.${namespace}"
  - "${cluster_name}-pd.${namespace}.svc"
  - "${cluster_name}-pd-peer"
  - "${cluster_name}-pd-peer.${namespace}"
  - "${cluster_name}-pd-peer.${namespace}.svc"
  - "*.${cluster_name}-pd-peer"
  - "*.${cluster_name}-pd-peer.${namespace}"
  - "*.${cluster_name}-pd-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

   `${cluster_name}` is the name of the cluster. Configure the items as follows:

  - Set `spec.secretName` to `${cluster_name}-pd-cluster-secret`.
  - Add `server auth` and `client auth` in `usages`.
  - Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
    * `${cluster_name}-pd`
    * `${cluster_name}-pd.${namespace}`
    * `${cluster_name}-pd.${namespace}.svc`
    * `${cluster_name}-pd-peer`
    * `${cluster_name}-pd-peer.${namespace}`
    * `${cluster_name}-pd-peer.${namespace}.svc`
    * `*.${cluster_name}-pd-peer`
    * `*.${cluster_name}-pd-peer.${namespace}`
    * `*.${cluster_name}-pd-peer.${namespace}.svc`
  - Add the following two IPs in `ipAddresses`. You can also add other IPs according to your needs:
    * `127.0.0.1`
    * `::1`
  - Add the Issuer created above in `issuerRef`.
  - For other attributes, refer to cert-manager API.

  After the object is created, `cert-manager` generates a `${cluster_name}-pd-` ↪ `cluster-secret` Secret object to be used by the PD component of the TiDB server.

- TiKV

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tikv-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tikv-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-tikv"
  - "${cluster_name}-tikv.${namespace}"
  - "${cluster_name}-tikv.${namespace}.svc"
  - "${cluster_name}-tikv-peer"
```

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

`${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${cluster_name}-tikv-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  * `${cluster_name}-tikv`
  * `${cluster_name}-tikv.${namespace}`
  * `${cluster_name}-tikv.${namespace}.svc`
  * `${cluster_name}-tikv-peer`
  * `${cluster_name}-tikv-peer.${namespace}`
  * `${cluster_name}-tikv-peer.${namespace}.svc`
  * `*.${cluster_name}-tikv-peer`
  * `*.${cluster_name}-tikv-peer.${namespace}`
  * `*.${cluster_name}-tikv-peer.${namespace}.svc`
- Add the following 2 IPs in `ipAddresses`. You can also add other IPs according to your needs:
  * `127.0.0.1`
  * `::1`
- Add the Issuer created above in `issuerRef`.
- For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${cluster_name}-tikv` ↪ `-cluster-secret` Secret object to be used by the TiKV component of the TiDB server.

- TiDB

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-cluster-secret
  namespace: ${namespace}
```

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

`${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${cluster_name}-tidb-cluster-secret`
- Add `server auth` and `client auth` in `usages`
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  * `${cluster_name}-tidb`
  * `${cluster_name}-tidb.${namespace}`
  * `${cluster_name}-tidb.${namespace}.svc`
  * `${cluster_name}-tidb-peer`
  * `${cluster_name}-tidb-peer.${namespace}`
  * `${cluster_name}-tidb-peer.${namespace}.svc`
  * `*.${cluster_name}-tidb-peer`
  * `*.${cluster_name}-tidb-peer.${namespace}`
  * `*.${cluster_name}-tidb-peer.${namespace}.svc`
- Add the following 2 IPs in `ipAddresses`. You can also add other IPs according to your needs:

    * 127.0.0.1
    * ::1
  - Add the Issuer created above in `issuerRef`.
  - For other attributes, refer to cert-manager API.

  After the object is created, `cert-manager` generates a `${cluster_name}-tidb-cluster-secret` Secret object to be used by the TiDB component of the TiDB server.

- Pump

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-pump-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-pump-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${cluster_name}-pump"
  - "*.${cluster_name}-pump.${namespace}"
  - "*.${cluster_name}-pump.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

  `${cluster_name}` is the name of the cluster. Configure the items as follows:

  - Set `spec.secretName` to `${cluster_name}-pump-cluster-secret`
  - Add `server auth` and `client auth` in `usages`
  - Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
    * *.${cluster_name}-pump
    * *.${cluster_name}-pump.${namespace}
    * *.${cluster_name}-pump.${namespace}.svc

  - Add the following 2 IPs in `ipAddresses`. You can also add other IPs according to your needs:
    * `127.0.0.1`
    * `::1`
  - Add the Issuer created above in the `issuerRef`
  - For other attributes, refer to cert-manager API.

  After the object is created, `cert-manager` generates a `${cluster_name}-pump -cluster-secret` Secret object to be used by the Pump component of the TiDB server.

- Drainer

  Drainer is deployed using Helm. The `dnsNames` field varies with different configuration of the `values.yaml` file.

  If you set the `drainerName` attributes when deploying Drainer as follows:

```
...
# Changes the name of the statefulset and Pod.
# The default value is clusterName-ReleaseName-drainer
# Does not change the name of an existing running Drainer, which is unsupported.
drainerName: my-drainer
...
```

  Then you need to configure the certificate as described below:

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-drainer-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-drainer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${drainer_name}"
  - "*.${drainer_name}.${namespace}"
  - "*.${drainer_name}.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
```

```
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

If you didn't set the `drainerName` attribute when deploying Drainer, configure the `dnsNames` attributes as follows:

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-drainer-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-drainer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${cluster_name}-${release_name}-drainer"
  - "*.${cluster_name}-${release_name}-drainer.${namespace}"
  - "*.${cluster_name}-${release_name}-drainer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

`${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the TiDB cluster is deployed. `${release_name}` is the `release name` you set when `helm install` is executed. `${drainer_name}` is `drainerName` in the `values.yaml` file. You can also add your customized `dnsNames`.

- Set `spec.secretName` to `${cluster_name}-drainer-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- See the above descriptions for `dnsNames`.
- Add the following 2 IPs in `ipAddresses`. You can also add other IPs according to your needs:

    * 127.0.0.1
    * ::1
  - Add the Issuer created above in `issuerRef`.
  - For other attributes, refer to cert-manager API.

  After the object is created, `cert-manager` generates a `${cluster_name}-drainer -cluster-secret` Secret object to be used by the Drainer component of the TiDB server.

- TiCDC

  Starting from v4.0.3, TiCDC supports TLS. TiDB Operator supports enabling TLS for TiCDC since v1.1.3.

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-ticdc-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-ticdc-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-ticdc"
  - "${cluster_name}-ticdc.${namespace}"
  - "${cluster_name}-ticdc.${namespace}.svc"
  - "${cluster_name}-ticdc-peer"
  - "${cluster_name}-ticdc-peer.${namespace}"
  - "${cluster_name}-ticdc-peer.${namespace}.svc"
  - "*.${cluster_name}-ticdc-peer"
  - "*.${cluster_name}-ticdc-peer.${namespace}"
  - "*.${cluster_name}-ticdc-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

  In the file, `${cluster_name}` is the name of the cluster:

  - Set `spec.secretName` to `${cluster_name}-ticdc-cluster-secret`.
  - Add `server auth` and `client auth` in `usages`.
  - Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
    * `${cluster_name}-ticdc`
    * `${cluster_name}-ticdc.${namespace}`
    * `${cluster_name}-ticdc.${namespace}.svc`
    * `${cluster_name}-ticdc-peer`
    * `${cluster_name}-ticdc-peer.${namespace}`
    * `${cluster_name}-ticdc-peer.${namespace}.svc`
    * `*.${cluster_name}-ticdc-peer`
    * `*.${cluster_name}-ticdc-peer.${namespace}`
    * `*.${cluster_name}-ticdc-peer.${namespace}.svc`
  - Add the following 2 IPs in `ipAddresses`. You can also add other IPs according to your needs:
    * `127.0.0.1`
    * `::1`
  - Add the Issuer created above in `issuerRef`.
  - For other attributes, refer to cert-manager API.

  After the object is created, `cert-manager` generates a `${cluster_name}-ticdc -cluster-secret` Secret object to be used by the TiCDC component of the TiDB server.

- TiFlash

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-tiflash-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tiflash-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-tiflash"
  - "${cluster_name}-tiflash.${namespace}"
  - "${cluster_name}-tiflash.${namespace}.svc"
```

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

In the file, `${cluster_name}` is the name of the cluster:

- Set `spec.secretName` to `${cluster_name}-tiflash-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  * `${cluster_name}-tiflash`
  * `${cluster_name}-tiflash.${namespace}`
  * `${cluster_name}-tiflash.${namespace}.svc`
  * `${cluster_name}-tiflash-peer`
  * `${cluster_name}-tiflash-peer.${namespace}`
  * `${cluster_name}-tiflash-peer.${namespace}.svc`
  * `*.${cluster_name}-tiflash-peer`
  * `*.${cluster_name}-tiflash-peer.${namespace}`
  * `*.${cluster_name}-tiflash-peer.${namespace}.svc`
- Add the following 2 IP addresses in `ipAddresses`. You can also add other IP addresses according to your needs:
  * `127.0.0.1`
  * `::1`
- Add the Issuer created above in `issuerRef`.
- For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${cluster_name}-tiflash -cluster-secret` Secret object to be used by the TiFlash component of the TiDB server.

- TiKV Importer

  If you need to restore data using TiDB Lightning, you need to generate a server-side certificate for the TiKV Importer component.

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
```

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

In the file, `${cluster_name}` is the name of the cluster:

- Set `spec.secretName` to `${cluster_name}-importer-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  * `${cluster_name}-importer`
  * `${cluster_name}-importer.${namespace}`
  * `${cluster_name}-importer.${namespace}.svc`
- Add the following 2 IP addresses in `ipAddresses`. You can also add other IP addresses according to your needs:
  * `127.0.0.1`
  * `::1`
- Add the Issuer created above in `issuerRef`.
- For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${cluster_name}-importer-cluster-secret` Secret object to be used by the TiKV Importer

component of the TiDB server.

- TiDB Lightning

  If you need to restore data using TiDB Lightning, you need to generate a server-side certificate for the TiDB Lightning component.

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
  name: ${cluster_name}-lightning-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-lightning-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  organization:
  - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${cluster_name}-lightning"
  - "${cluster_name}-lightning.${namespace}"
  - "${cluster_name}-lightning.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

  In the file, `${cluster_name}` is the name of the cluster:

  - Set `spec.secretName` to `${cluster_name}-lightning-cluster-secret`.
  - Add `server auth` and `client auth` in `usages`.
  - Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
    * `${cluster_name}-lightning`
    * `${cluster_name}-lightning.${namespace}`
    * `${cluster_name}-lightning.${namespace}.svc`
  - Add the following 2 IP addresses in `ipAddresses`. You can also add other IP addresses according to your needs:
    * `127.0.0.1`
    * `::1`

- Add the Issuer created above in `issuerRef`.
- For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${cluster_name}-lightning-cluster-secret` Secret object to be used by the TiDB Lightning component of the TiDB server.

4. Generate the client-side certificate for components of the TiDB cluster.

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
metadata:
    name: ${cluster_name}-cluster-client-secret
    namespace: ${namespace}
spec:
    secretName: ${cluster_name}-cluster-client-secret
    duration: 8760h # 365d
    renewBefore: 360h # 15d
    organization:
    - PingCAP
    commonName: "TiDB"
    usages:
    - client auth
    issuerRef:
    name: ${cluster_name}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
```

`${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${cluster_name}-cluster-client-secret`.
- Add `client auth` in `usages`.
- You can leave `dnsNames` and `ipAddresses` empty.
- Add the Issuer created above in `issuerRef`.
- For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${cluster_name}-cluster-client-secret` Secret object to be used by the clients of the TiDB components.

5.2.2 Deploy the TiDB cluster

When you deploy a TiDB cluster, you can enable TLS between TiDB components, and set the `cert-allowed-cn` configuration item (for TiDB, the configuration item is `cluster-verify-cn`) to verify the CN (Common Name) of each component's certificate.

Note:

Currently, you can set only one value for the `cert-allowed-cn` configuration item of PD. Therefore, the `commonName` of all `Certificate` objects must be the same.

In this step, you need to perform the following operations:

- Create a TiDB cluster
- Enable TLS between the TiDB components, and enable CN verification
- Deploy a monitoring system
- Deploy the Pump component, and enable CN verification

1. Create a TiDB cluster:

   Create the `tidb-cluster.yaml` file:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
 name: ${cluster_name}
 namespace: ${namespace}
spec:
 tlsCluster:
   enabled: true
 version: v5.0.6
 timezone: UTC
 pvReclaimPolicy: Retain
 pd:
   baseImage: pingcap/pd
   replicas: 1
   requests:
     storage: "1Gi"
   config:
     security:
       cert-allowed-cn:
         - TiDB
 tikv:
   baseImage: pingcap/tikv
   replicas: 1
   requests:
     storage: "1Gi"
   config:
```

```
    security:
      cert-allowed-cn:
        - TiDB
 tidb:
   baseImage: pingcap/tidb
   replicas: 1
   service:
     type: ClusterIP
   config:
     security:
       cluster-verify-cn:
         - TiDB
 pump:
   baseImage: pingcap/tidb-binlog
   replicas: 1
   requests:
     storage: "1Gi"
   config:
     security:
       cert-allowed-cn:
         - TiDB
---
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
 name: ${cluster_name}
 namespace: ${namespace}
spec:
 clusters:
 - name: ${cluster_name}
 prometheus:
   baseImage: prom/prometheus
   version: v2.11.1
 grafana:
   baseImage: grafana/grafana
   version: 6.0.1
 initializer:
   baseImage: pingcap/tidb-monitor-initializer
   version: v5.0.6
 reloader:
   baseImage: pingcap/tidb-monitor-reloader
   version: v1.0.1
 imagePullPolicy: IfNotPresent
```

Execute `kubectl apply -f tidb-cluster.yaml` to create a TiDB cluster.

This operation also includes deploying a monitoring system and the Pump component.

2. Create a Drainer component and enable TLS and CN verification:

    - Method 1: Set `drainerName` when you create Drainer.
      Edit the `values.yaml` file, set `drainer-name`, and enable the TLS feature:

      ```
      ...
      drainerName: ${drainer_name}
      tlsCluster:
        enabled: true
        certAllowedCN:
          - TiDB
      ...
      ```

      Deploy the Drainer cluster:

      ```
      helm install ${release_name} pingcap/tidb-drainer --namespace=${
          ↪ namespace} --version=${helm_version} -f values.yaml
      ```

    - Method 2: Do not set `drainerName` when you create Drainer.
      Edit the `values.yaml` file, and enable the TLS feature:

      ```
      ...
      tlsCluster:
        enabled: true
        certAllowedCN:
          - TiDB
      ...
      ```

      Deploy the Drainer cluster:

      ```
      helm install ${release_name} pingcap/tidb-drainer --namespace=${
          ↪ namespace} --version=${helm_version} -f values.yaml
      ```

3. Create the Backup/Restore resource object:

    - Create the `backup.yaml` file:

      ```
      apiVersion: pingcap.com/v1alpha1
      kind: Backup
      metadata:
        name: ${cluster_name}-backup
        namespace: ${namespace}
      spec:
        backupType: full
        br:
          cluster: ${cluster_name}
      ```

```
    clusterNamespace: ${namespace}
    sendCredToTikv: true
  from:
    host: ${host}
    secretName: ${tidb_secret}
    port: 4000
    user: root
  s3:
    provider: aws
    region: ${my_region}
    secretName: ${s3_secret}
    bucket: ${my_bucket}
    prefix: ${my_folder}
```

Deploy Backup:

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

- Create the `restore.yaml` file:

```
yaml apiVersion: pingcap.com/v1alpha1 kind: Restore metadata:
↪ name: ${cluster_name}-restore namespace: ${namespace} spec:
↪ backupType: full br: cluster: ${cluster_name} clusterNamespace:
↪  ${namespace} sendCredToTikv: true to: host: ${host} secretName
↪ : ${tidb_secret} port: 4000 user: root s3:   provider: aws
↪   region: ${my_region}  secretName: ${s3_secret} bucket: ${
↪ my_bucket}  prefix: ${my_folder}
```

Deploy Restore:

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

5.2.3 Configure `pd-ctl`, `tikv-ctl` and connect to the cluster

1. Mount the certificates.

   Configure `spec.pd.mountClusterClientSecret: true` and `spec.tikv.mountClusterClientSecre`
   `↪ : true` with the following command:

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

Note:

- The above configuration will trigger the rolling update of PD and TiKV cluster.
- The above configurations are supported since TiDB Operator v1.1.5.

2. Use `pd-ctl` to connect to the PD cluster.

   Get into the PD Pod:

   ```
   kubectl exec -it ${cluster_name}-pd-0 -n ${namespace} sh
   ```

   Use `pd-ctl`:

   ```
   cd /var/lib/cluster-client-tls
   /pd-ctl --cacert=ca.crt --cert=tls.crt --key=tls.key -u https
      ↪ ://127.0.0.1:2379 member
   ```

3. Use `tikv-ctl` to connect to the TiKV cluster.

   Get into the TiKV Pod:

   ```
   kubectl exec -it ${cluster_name}-tikv-0 -n ${namespace} sh
   ```

   Use `tikv-ctl`:

   ```
   cd /var/lib/cluster-client-tls
   /tikv-ctl --ca-path=ca.crt --cert-path=tls.crt --key-path=tls.key --
      ↪ host 127.0.0.1:20160 cluster
   ```

5.3 Renew and Replace the TLS Certificate

This document introduces how to renew and replace certificates of the corresponding components before certificates expire, taking TLS certificates between PD, TiKV, and TiDB components in the TiDB cluster as an example.

If you need to renew and replace certificates between other components in the TiDB cluster, TiDB server-side certificate, or MySQL client-side certificate, you can take similar steps to complete the operation.

The renewal and replacement operations in this document assume that the original certificates have not expired. If the original certificates expire or become invalid, to generate new certificates and restart the TiDB cluster, refer to Enable TLS between TiDB components or Enable TLS for MySQL client.

5.3.1 Renew and replace certificates issued by the `cfssl` system

If the original TLS certificates are issued by the `cfssl` system and the original certificates have not expired, you can renew and replace the certificates between PD, TiKV and TiDB components as follows.

5.3.1.1 Renew and replace the CA certificate

> **Note:**
>
> If you don't need to renew the CA certificate, you can skip the operations in this section and directly refer to renew and replace certificates between components.

1. Back up the original CA certificate and key.

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

2. Generate the new CA certificate and key based on the configuration of the original CA certificate and certificate signing request (CSR).

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

> **Note:**
>
> If necessary, you can update `expiry` in the configuration file and in CSR.

3. Back up the new CA certificate and key, and generate a combined CA certificate based on the original CA certificate and the new CA certificate.

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

4. Update each corresponding Kubernetes Secret objects based on the combined CA certificate.

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

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

In the above command, `${cluster_name}` is the name of the cluster, and `${namespace ↪ }` is the namespace in which the TiDB cluster is deployed.

> **Note:**
>
> The above command only renews the server-side CA certificate and the client-side CA certificate between PD, TiKV, and TiDB components. If you need to renew the server-side CA certificates for other components, such as TiCDC and TiFlash, you can execute the similar command.

5. Perform the rolling restart to components that need to load the combined CA certificate.

   After the completion of the rolling restart, based on the combined CA certificate, each component can accept the certificate issued by either the original CA certificate or the new CA certificate at the same time.

5.3.1.2 Renew and replace certificates between components

> **Note:**
>
> Before renewing and replacing certificates between components, make sure that the CA certificate can verify the certificates between components before and after the renewal as valid. If you have renewed and replaced the CA certificate, make sure that the TiDB cluster is restarted based on the new CA certificate.

1. Generate new server-side and client-side certificates based on the original configuration information of each component.

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

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

Note:

- The above command assumes that you have renewed and replaced the CA certificate and saved the new CA certificate as `ca.new.pem` and the new key as `ca-key.new.pem`. If you have not renewed the CA certificate and the key, modify the corresponding parameters in the command to `ca.pem` and `ca-key.pem`.
- The above command only generates the server-side and the client-side certificates between PD, TiKV, and TiDB components. If you need to generate the server-side CA certificates for other components, such as TiCDC and TiFlash, you can execute the similar command.

2. Update each corresponding Kubernetes Secret object based on the newly generated server-side and client-side certificates.

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

In the above command, `${cluster_name}` is the name of the cluster, and `${namespace ↪ }` is the namespace in which the TiDB cluster is deployed.

> **Note:**
>
> The above command only renews the server-side and the client-side certificate between PD, TiKV, and TiDB components. If you need to renew the server-side certificates for other components, such as TiCDC and TiFlash, you can execute the similar command.

3. Perform the rolling restart to components that need to load the new certificates.

   After the completion of the rolling restart, each component use the new certificate for TLS communication. If you refer to Renew and replace the CA certificate and make each component load the combined CA certificate, each component can still accept the certificate issued by the original CA certificate.

5.3.1.3 Optional: Remove the original CA certificate from the combined CA certificate

After you renew and replace the combined CA certificate, server-side and client-side certificates between components, you might want to remove the original CA certificate (for example, because the CA certificate has expired or the private key is compromised). To remove the original CA certificate, take steps as follows:

1. Renew the Kubernetes Secret objects based on the new CA certificate.

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

   In the above command, `${cluster_name}` is the name of the cluster, and `${namespace ↪ }` is the namespace in which the TiDB cluster is deployed.

Note:

- The above command assumes that you have renewed and replaced the CA certificate and saved the new CA certificate as `ca.new.pem`.

2. Perform the rolling restart to components that need to load the new certificates.

   After the completion of the rolling restart, each component can only accept the certificate issued by the new CA certificate.

5.3.2 Renew and replace the certificate issued by `cert-manager`

If the original TLS certificate is issued by the `cert-manager` system, and the original certificate has not expired, the procedure varies with whether to renew the CA certificate.

5.3.2.1 Renew and replace the CA certificate and certificates between components

When you use `cert-manager` to issue the certificate, if you specify the `spec.renewBefore` of the `Certificate` resource, `cert-manager` can automatically update the certificate before it expires.

Although `cert-manager` can automatically renew the CA certificate and the corresponding Kubernetes Secret objects, it currently does not support merging the old and new CA certificates into a combined CA certificate to accept certificates issued by the new and old CA certificates at the same time. Therefore, during the process of renewing and replacing the CA certificate, the cluster components cannot authenticate each other via TLS.

Warning:

Because the components cannot accept certificates issued by the new and old CAs at the same time, during the process of renewing and replacing certificates, some components' Pods need to be recreated. This might cause some requests to access the TiDB cluster to fail.

The steps to renew and replace the CA certificates of PD, TiKV, TiDB and certificates between components are as follows.

1. The `cert-manager` automatically renews the CA certificate and the Kubernetes Secret object `${cluster_name}-ca-secret` before the certificate expires.

   `${cluster_name}` is the name of the cluster.

To manually renew the CA certificate, you can directly delete the corresponding Kubernetes Secret objects and trigger `cert-manager` to regenerate the CA certificate.

2. Delete the Kubernetes Secret objects corresponding to the certificate of each component.

```
kubectl delete secret ${cluster_name}-pd-cluster-secret --namespace=${
    ↪ namespace}
kubectl delete secret ${cluster_name}-tikv-cluster-secret --namespace=$
    ↪ {namespace}
kubectl delete secret ${cluster_name}-tidb-cluster-secret --namespace=$
    ↪ {namespace}
kubectl delete secret ${cluster_name}-cluster-client-secret --namespace
    ↪ =${namespace}
```

In the above command, `${cluster_name}` is the name of the cluster, and `${namespace ↪ }` is the namespace in which the TiDB cluster is deployed.

3. Wait for `cert-manager` to issue new certificates for each component based on the new CA certificate.

Observe the output of `kubectl get secret --namespace=${namespace}` until the Kubernetes Secret objects corresponding to all components are created.

4. Forcibly recreate the Pods of the PD, TiKV, and TiDB components in sequence.

Because `cert-manager` does not support combined CA certificates, if you try to perform a rolling update of each component, the Pods using the different CAs to issue certificates cannot communicate with each other via TLS. Therefore, you need to delete the Pods forcibly and recreate the Pods based on the certificate issued by the new CA.

```
kubectl delete -n ${namespace} pod ${pod_name}
```

In the above command, `${namespace}` is the namespace in which the TiDB cluster is deployed, and `${pod_name}` is the Pod name of each replica of PD, TiKV, and TiDB.

5.3.2.2 Only renew and replace certificates between components

1. The `cert-manager` automatically updates the certificate of each component and the Kubernetes Secret object before the certificate expires.

For PD, TiKV, and TiDB components, the namespace in which the TiDB cluster is deployed contains the following Kubernetes Secret objects:

```
${cluster_name}-pd-cluster-secret
${cluster_name}-tikv-cluster-secret
${cluster_name}-tidb-cluster-secret
${cluster_name}-cluster-client-secret
```

In the above command, `${cluster_name}` is the name of the cluster.

If you want to manually update the certificate between components, you can directly delete the corresponding Kubernetes Secret objects and trigger `cert-manager` to re-generate the certificate between components.

2. For certificates between components, each component automatically reloads the new certificates when creating the new connection later.

Note:

- Currently, each component does not support reload CA certificates manually, you need to refer to renew and replace the CA certificate and certificates between components.
- For the TiDB server-side certificate, you can manually reload by referring to any of the following methods:
  - Refer to Reload certificate, key, and CA.
  - Refer to Rolling restart the TiDB Cluster to perform a rolling restart of TiDB server.

6 Operate

6.1 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.

6.1.1 Upgrade using TidbCluster CR

If the TiDB cluster is deployed directly using TidbCluster CR, or deployed using Helm but switched to management by TidbCluster CR, you can upgrade the TiDB cluster by the following steps.

6.1.1.1 Upgrade the version of TiDB using TidbCluster CR

1. Modify the image configurations of all components in TidbCluster CR.

   Usually, components in a cluster are in the same version. You can upgrade the TiDB cluster simply by modifying `spec.version`. If you need to use different versions for different components, configure `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`.

   The `version` field has following formats:

   - `spec.version`: the format is `imageTag`, such as `v5.0.6`
   - `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`: the format is `imageTag`, such as `v3.1.0`

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

2. Check the upgrade progress:

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

   After all the Pods finish rebuilding and become `Running`, the upgrade is completed.

6.1.1.2 Force an upgrade of TiDB cluster using TidbCluster CR

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` to force an upgrade of the cluster to recover cluster functionality.

First, set `annotation` for the cluster:

```
kubectl annotate --overwrite tc ${cluster_name} -n ${namespace} tidb.pingcap
    ↪ .com/force-upgrade=true
```

Change the related PD configuration to make sure that PD is in a normal state.

Note:

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 ${cluster_name} -n ${namespace} tidb.
    ↪ pingcap.com/force-upgrade-
```

6.1.1.3 Modify TiDB cluster configuration

Note:

- After the cluster is started for the first time, some PD configuration items are persisted in etcd. The persisted configuration in etcd takes precedence over that in PD. Therefore, after the first start, you cannot modify some PD configuration using parameters. You need to dynamically modify the configuration using SQL statements, pd-ctl, or PD server API. Currently, among all the configuration items listed in Modify PD configuration online, except `log.level`, all the other configuration items cannot be modified using parameters after the first start. If you modify configuration items by dynamically modifying configuration online, after rolling upgrade, the configuration might be overwritten by CR.

1. Refer to Configure TiDB components to modify the component configuration in the `TidbCluster` CR of the cluster:

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

2. View the updating progress after the configuration is modified:

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

After all the Pods are recreated and are in the `Running` state, the configuration is successfully modified.

Note:

By default, TiDB (starting from v4.0.2) periodically shares usage details with PingCAP to help understand how to improve the product. For details about what is shared and how to disable the sharing, see Telemetry.

6.2 Upgrade TiDB Operator and Kubernetes

This document describes how to upgrade TiDB Operator and Kubernetes.

Note:

You can check the currently supported versions of TiDB Operator using the `helm search repo -l tidb-operator` command. If the command output does not include the latest version, update the repo using the `helm repo update` command. For details, refer to Configure the Help repo.

6.2.1 Upgrade TiDB Operator online

1. Update CustomResourceDefinition (CRD) for Kubernetes. For more information about CRD, see CustomResourceDefinition.

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

2. Get the `values.yaml` file of the `tidb-operator` chart for the new TiDB Operator version.

```
mkdir -p ${HOME}/tidb-operator/v1.1.15 && \
helm inspect values pingcap/tidb-operator --version=v1.1.15 > ${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml -n tidb-admin
```

3. In the `${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml` file, modify the `operatorImage` version to the new TiDB Operator version. Merge the customized configuration in the old `values.yaml` file to the `${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml` file, and then execute `helm upgrade`:

```
helm upgrade tidb-operator pingcap/tidb-operator --version=v1.1.15 -f ${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml
```

After all the Pods start normally, execute the following command to check the image of TiDB Operator:

```
kubectl get po -n tidb-admin -l app.kubernetes.io/instance=tidb-operator -o yaml | grep 'image:.*operator:'
```

If TiDB Operator is successfully upgraded, the expected output is as follows. `v1.1.15` represents the desired version of TiDB Operator.

```
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
```

Note:

After TiDB Operator is upgraded, the `discovery` deployment in all TiDB clusters will be automatically upgraded to the corresponding version of TiDB Operator.

6.2.2 Upgrade TiDB Operator offline

If your server cannot access the Internet, you can take the following steps to upgrade TiDB Operator offline:

1. Download the files and images required for the upgrade using a machine with the Internet access:

    1. Download the `crd.yaml` file for the new TiDB Operator version. For more information about CRD, see CustomResourceDefinition.

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

    2. Download the `tidb-operator` chart package file.

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

    3. Download the Docker images required for the new TiDB Operator version:

    ```
    docker pull pingcap/tidb-operator:v1.1.15
    docker pull pingcap/tidb-backup-manager:v1.1.15

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

2. Upload the downloaded files and images to the server that needs to be upgraded, and then take the following steps for installation:

    1. Install the `crd.yaml` file for TiDB Operator:

    ```
    kubectl apply -f . /crd.yaml
    ```

    2. Unpack the `tidb-operator` chart package file, and then copy the `values.yaml` file:

```
tar zxvf tidb-operator-v1.1.15.tgz && \
mkdir -p ${HOME}/tidb-operator/v1.1.15 &&
cp tidb-operator/values.yaml ${HOME}/tidb-operator/v1.1.15/values-
    ↪ tidb-operator.yaml
```

3. Install the Docker images on the server:

```
docker load -i tidb-operator-v1.1.15.tar
docker load -i tidb-backup-manager-v1.1.15.tar
```

3. In the `${HOME}/tidb-operator/v1.1.15/values-tidb-operator.yaml` file, modify the `operatorImage` version to the new TiDB Operator version. Merge the customized configuration in the old `values.yaml` file to the `${HOME}/tidb-operator/v1.1.15/` ↪ `values-tidb-operator.yaml` file, and then execute `helm upgrade`:

```
helm upgrade tidb-operator ./tidb-operator --version=v1.1.15 -f ${HOME
    ↪ }/tidb-operator/v1.1.15/values-tidb-operator.yaml
```

After all the Pods start normally, execute the following command to check the image version of TiDB Operator:

```
```shell
kubectl get po -n tidb-admin -l app.kubernetes.io/instance=tidb-operator -o
    ↪ yaml | grep 'image:.*operator:'
```
```

If TiDB Operator is successfully upgraded, the expected output is as follows. `v1.1.15` represents the new version of TiDB Operator.

```
```
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
image: pingcap/tidb-operator:v1.1.15
image: docker.io/pingcap/tidb-operator:v1.1.15
```
```

Note:

After TiDB Operator is upgraded, the `discovery` deployment in all TiDB clusters will be automatically upgraded to the specified version of TiDB Operator.

6.2.3 Upgrade TiDB Operator from v1.0 to v1.1 or later releases

Since TiDB Operator v1.1.0, PingCAP no longer updates or maintains the tidb-cluster chart. The components and features that have been managed using the tidb-cluster chart will be managed by CR (Custom Resource) or dedicated charts in v1.1. For more details, refer to TiDB Operator v1.1 Notes.

6.3 Perform a Canary Upgrade on TiDB Operator

This document describes how to perform a canary upgrade on TiDB Operator. Using canary upgrades, you can prevent normal TiDB Operator upgrade from causing an unexpected impact on all the TiDB clusters in Kubernetes. After you confirm the impact of TiDB Operator upgrade or that the upgraded TiDB Operator works stably, you can normally upgrade TiDB Operator.

Note:

- You can perform a canary upgrade only on `tidb-controller-manager` and `tidb-scheduler`. AdvancedStatefulSet controller and `tidb-admission-webhook` do not support the canary upgrade.
- Canary upgrade is supported since v1.1.10. The version of your current TiDB Operator should be >= v1.1.10.

6.3.1 Related parameters

To support canary upgrade, some parameters are added to the `values.yaml` file in the `tidb-operator` chart. See Related parameters for details.

6.3.2 Canary upgrade process

1. Configure selector for the current TiDB Operator:

   Refer to Upgrade TiDB Operator. Add the following configuration in the `values.yaml` file, and upgrade TiDB Operator:

   ```
   controllerManager:
     selector:
     - version!=canary
   ```

   If you have already performed the step above, skip to Step 2.

2. Deploy the canary TiDB Operator:

   Refer to Deploy TiDB Operator. Add the following configuration in the `values.yaml` file, and deploy the canary TiDB Operator in **a different namespace** (such as `tidb-admin-canary`) with a **different Helm Release Name** (such as `helm install tidb-operator-canary ...`):

```
controllerManager:
  selector:
  - version=canary
appendReleaseSuffix: true
#scheduler:
#  create: false
advancedStatefulset:
  create: false
admissionWebhook:
  create: false
```

   Note:

   - It is recommended to deploy the new TiDB Operator in a separate namespace.
   - Set `appendReleaseSuffix` to `true`.
   - If you do not need to perform a canary upgrade on `tidb-scheduler`, configure `scheduler.create: false`.
   - If you configure `scheduler.create: true`, a scheduler named `{{ .scheduler.schedulerName }}-{{.Release.Name}}` will be created. To use this scheduler, configure `spec.schedulerName` in the `TidbCluster` CR to the name of this scheduler.
   - You need to set `advancedStatefulset.create: false` and `admissionWebhook.create: false`, because AdvancedStatefulSet controller and `tidb-admission-webhook` do not support the canary upgrade.

3. To test the canary upgrade of `tidb-controller-manager`, set labels for a TiDB cluster by running the following command:

```
kubectl -n ${namespace} label tc ${cluster_name} version=canary
```

   Check the logs of the two deployed `tidb-controller-manager`s, and you can see this TiDB cluster is now managed by the canary TiDB Operator:

   1. View the log of `tidb-controller-manager` of the current TiDB Operator:

```
kubectl -n tidb-admin logs tidb-controller-manager-55b887bdc9-lzdwv
```

```
I0305 07:52:04.558973    1 tidb_cluster_controller.go:148]
    ↪ TidbCluster has been deleted tidb-cluster-1/basic1
```

2. View the log of `tidb-controller-manager` of the canary TiDB Operator:

```
kubectl -n tidb-admin-canary logs tidb-controller-manager-canary-6
    ↪ dcb9bdd95-qf4qr
```

```
I0113 03:38:43.859387    1 tidbcluster_control.go:69] TidbCluster:
    ↪ [tidb-cluster-1/basic1] updated successfully
```

4. To test the canary upgrade of `tidb-scheduler`, modify `spec.schedulerName` of some TiDB cluster to `tidb-scheduler-canary` by running the following command:

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

After the modification, all components in the cluster will be rolling updated.

Check the logs of `tidb-scheduler` of the canary TiDB Operator, and you can see this TiDB cluster is now using the canary `tidb-scheduler`:

```
kubectl -n tidb-admin-canary logs tidb-scheduler-canary-7f7b6c7c6-j5p2j
    ↪  -c tidb-scheduler
```

5. After the tests, you can revert the changes in Step 3 and Step 4 so that the TiDB cluster is again managed by the current TiDB Operator.

```
kubectl -n ${namespace} label tc ${cluster_name} version-
```

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

6. Delete the canary TiDB Operator:

```
helm -n tidb-admin-canary uninstall ${release_name}
```

7. Refer to Upgrade TiDB Operator and upgrade the current TiDB Operator normally.

6.4 Pause Sync of a TiDB Cluster in Kubernetes

This document introduces how to pause sync of a TiDB cluster in Kubernetes using configuration.

6.4.1 What is sync in TiDB Operator

In TiDB Operator, controller regulates the state of the TiDB cluster in Kubernetes. The controller constantly compares the desired state recorded in the `TidbCluster` object with the actual state of the TiDB cluster. This process is referred to as **sync** generally. For more details, refer to TiDB Operator Architecture.

6.4.2 Use scenarios

Here are some cases where you might need to pause sync of a TiDB cluster in Kubernetes.

- Avoid unexpected rolling update

  To prevent new versions of TiDB Operator from introducing compatibility issues into the clusters, before updating TiDB Operator, you can pause sync of TiDB clusters. After updating TiDB Operator, you can resume syncing clusters one by one, or specify a time for resume. In this way, you can observe how the rolling update of TiDB Operator would affect the cluster.

- Avoid multiple rolling restarts

  In some cases, you might need to continuously modify the cluster over a period of time, but do not want to restart the TiDB cluster many times. To avoid multiple rolling restarts, you can pause sync of the cluster. During the sync pausing, any change of the configuration does not take effect on the cluster. After you finish the modification, resume sync of the TiDB cluster. All changes can be applied in a single rolling restart.

- Maintenance window

  In some situations, you can update or restart the TiDB cluster only during a maintenance window. When outside the maintenance window, you can pause sync of the TiDB cluster, so that any modification to the specs does not take effect. When inside the maintenance window, you can resume sync of the TiDB cluster to allow TiDB cluster to rolling update or restart.

6.4.3 Pause sync

1. Execute the following command to edit the TiDB cluster configuration. `${cluster_name}` represents the name of the TiDB cluster, and `${namespace}` refers to the TiDB cluster namespace.

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

2. Configure `spec.paused: true` as follows, and save changes. The sync of TiDB cluster components (PD, TiKV, TiDB, TiFlash, TiCDC, Pump) is then paused.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  ...
spec:
  ...
  paused: true # Pausing sync of TiDB cluster
  pd:
    ...
```

```
  tikv:
    ...
  tidb:
    ...
```

3. To confirm the sync status of a TiDB cluster, execute the following command. `${pod_name}` is the name of tidb-controller-manager Pod, and `${namespace}` is the namespace of TiDB Operator.

```
kubectl logs ${pod_name} -n ${namespace} | grep paused
```

The expected output is as follows. The sync of all components in the TiDB cluster is paused.

```
I1207 11:09:59.029949    1 pd_member_manager.go:92] tidb cluster
    ↪ default/basic is paused, skip syncing for pd service
I1207 11:09:59.029977    1 pd_member_manager.go:136] tidb cluster
    ↪ default/basic is paused, skip syncing for pd headless service
I1207 11:09:59.035437    1 pd_member_manager.go:191] tidb cluster
    ↪ default/basic is paused, skip syncing for pd statefulset
I1207 11:09:59.035462    1 tikv_member_manager.go:116] tikv cluster
    ↪ default/basic is paused, skip syncing for tikv service
I1207 11:09:59.036855    1 tikv_member_manager.go:175] tikv cluster
    ↪ default/basic is paused, skip syncing for tikv statefulset
I1207 11:09:59.036886    1 tidb_member_manager.go:132] tidb cluster
    ↪ default/basic is paused, skip syncing for tidb headless service
I1207 11:09:59.036895    1 tidb_member_manager.go:258] tidb cluster
    ↪ default/basic is paused, skip syncing for tidb service
I1207 11:09:59.039358    1 tidb_member_manager.go:188] tidb cluster
    ↪ default/basic is paused, skip syncing for tidb statefulset
```

6.4.4 Resume sync

To resume the sync of the TiDB cluster, configure `spec.paused: false` in the TidbCluster CR.

1. Execute the following command to edit the TiDB cluster configuration. `${cluster_name}` represents the name of the TiDB cluster, and `${namespace}` refers to the TiDB cluster namespace.

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

2. Configure `spec.paused: false` as follows, and save changes. The sync of TiDB cluster components (PD, TiKV, TiDB, TiFlash, TiCDC, Pump) is then resumed.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  ...
spec:
  ...
  paused: false # Resuming sync of TiDB cluster
  pd:
    ...
  tikv:
    ...
  tidb:
    ...
```

3. To confirm the sync status of a TiDB cluster, execute the following command. `$ ↪ {pod_name}` represents the name of the tidb-controller-manager Pod, and `${namespace}` represents the namespace of TiDB Operator.

```
kubectl logs ${pod_name} -n ${namespace} | grep "Finished syncing
   ↪ TidbCluster"
```

The expected output is as follows. The `finished syncing` timestamp is later than the `paused` timestamp, which indicates that sync of the TiDB cluster has been resumed.

```
I1207 11:14:59.361353    1 tidb_cluster_controller.go:136] Finished
   ↪ syncing TidbCluster "default/basic" (368.816685ms)
I1207 11:15:28.982910    1 tidb_cluster_controller.go:136] Finished
   ↪ syncing TidbCluster "default/basic" (97.486818ms)
I1207 11:15:29.360446    1 tidb_cluster_controller.go:136] Finished
   ↪ syncing TidbCluster "default/basic" (377.51187ms)
```

6.5 Scale TiDB Cluster

6.5.1 Scale TiDB in Kubernetes

This document introduces how to horizontally and vertically scale a TiDB cluster in Kubernetes.

6.5.1.1 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.

Currently, the TiDB cluster supports management by TidbCluster Custom Resource (CR).

6.5.1.1.1 Scale PD, TiDB, and TiKV

Modify `spec.pd.replicas`, `spec.tidb.replicas`, and `spec.tikv.replicas` in the `TidbCluster` object of the cluster to a desired value using kubectl. You can modify the values in the local file or using online command.

- You can also online modify the `TidbCluster` definition in the Kubernetes cluster by running the following command:

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

Check whether the TiDB cluster in Kubernetes has updated to your desired definition by running the following command:

```
kubectl get tidbcluster ${cluster_name} -n ${namespace} -oyaml
```

In the `TidbCluster` file output by the command above, if the values of `spec.pd.` ↪ `replicas`, `spec.tidb.replicas`, and `spec.tikv.replicas` are consistent with the values you have modified, check whether the number of `TidbCluster` Pods has increased or decreased by running the following command:

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

For the PD and TiDB components, it might take 10-30 seconds to scale in or out.

For the TiKV component, it might take 3-5 minutes to scale in or out because the process involves data migration.

Scale out TiFlash

If TiFlash is deployed in the cluster, you can scale out TiFlash by modifying `spec.` ↪ `tiflash.replicas`.

Scale TiCDC

If TiCDC is deployed in the cluster, you can scale out TiCDC by modifying `spec.ticdc` ↪ `.replicas`.

Scale in TiFlash

1. Expose the PD service by using `port-forward`:

```
kubectl port-forward -n ${namespace} svc/${cluster_name}-pd 2379:2379
```

2. Open a **new** terminal tab or window. Check the maximum number (`N`) of replicas of all data tables with which TiFlash is enabled by running the following command:

```
curl 127.0.0.1:2379/pd/api/v1/config/rules/group/tiflash | grep count
```

   In the printed result, the largest value of `count` is the maximum number (`N`) of replicas of all data tables.

3. Go back to the terminal window in Step 1, where `port-forward` is running. Press Ctrl+C to stop `port-forward`.

4. After the scale-in operation, if the number of remaining Pods in TiFlash >= `N`, skip to Step 6. Otherwise, take the following steps:

   1. Refer to Access TiDB and connect to the TiDB service.
   2. For all the tables that have more replicas than the remaining Pods in TiFlash, run the following command:

```
alter table <db_name>.<table_name> set tiflash replica 0;
```

5. Wait for TiFlash replicas in the related tables to be deleted.

   Connect to the TiDB service, and run the following command:

```
SELECT * FROM information_schema.tiflash_replica WHERE TABLE_SCHEMA =
    ↪ '<db_name>' and TABLE_NAME = '<table_name>';
```

   If you cannot view the replication information of related tables, the TiFlash replicas are successfully deleted.

6. Modify `spec.tiflash.replicas` to scale in TiFlash.

   Check whether TiFlash in the TiDB cluster in Kubernetes has updated to your desired definition. Run the following command and see whether the value of `spec.tiflash.replicas` returned is expected:

```
kubectl get tidbcluster ${cluster-name} -n ${namespace} -oyaml
```

6.5.1.1.2 View the horizontal scaling status

To view the scaling status of the cluster, run the following command:

```
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, TiKV and TiFlash components do not trigger the rolling update operations during scaling in and out.
- When the TiKV component scales in, TiDB Operator 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 -n ${namespace} tidbcluster ${cluster_name} -o json | jq '.status.tikv.stores'`.
- When the number of `UP` stores is equal to or less than the parameter value of `MaxReplicas` in the PD configuration, the TiKV components can not be scaled in.
- 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.
- The TiFlash component has the same scale-in logic as TiKV.
- When the PD, TiKV, and TiFlash 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.

6.5.1.1.3 Horizontal scaling failure

During the horizontal scaling operation, Pods might go to the Pending state because of insufficient resources. See Troubleshoot the Pod in Pending state.

6.5.1.2 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.

Currently, the TiDB cluster supports management by TidbCluster Custom Resource (CR).

6.5.1.2.1 Vertical scaling operations

Modify `spec.pd.resources`, `spec.tikv.resources`, and `spec.tidb.resources` in the `TidbCluster` object that corresponds to the cluster to the desired values using kubectl.

If TiFlash is deployed in the cluster, you can scale up and down TiFlash by modifying `spec.tiflash.resources`.

If TiCDC is deployed in the cluster, you can scale up and down TiCDC by modifying `spec.ticdc.resources`.

6.5.1.2.2 View the vertical scaling progress

To view the upgrade progress of the cluster, run the following command:

```
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, and if PD, TiKV, and TiFlash use `Local PV`, they will 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.

6.5.1.2.3 Vertical scaling failure

During the vertical scaling operation, Pods might go to the Pending state because of insufficient resources. See Troubleshoot the Pod in Pending state for details.

6.5.2 Enable TidbCluster Auto-scaling

Kubernetes provides Horizontal Pod Autoscaler, a native API based on CPU utilization. Based on Kubernetes, TiDB 4.0 has implemented an elastic scheduling mechanism. Correspondingly, in TiDB Operator 1.1 and later versions, you can enable the auto-scaling feature to enable elastic scheduling. This document introduces how to enable and use the auto-scaling feature of `TidbCluster`.

6.5.2.1 Enable the auto-scaling feature

> **Warning:**
>
> - The auto-scaling feature is in the alpha stage. It is highly **not recommended** to enable this feature in the critical production environment.
> - It is recommended to try this feature in a test environment on the internal network. PingCAP welcomes your comments and suggestions to help improve this feature.

To turn this feature on, you need to enable some related configurations in TiDB Operator. The auto-scaling feature is disabled by default. Take the following steps to manually turn it on.

1. Edit the `values.yaml` file in TiDB Operator.

   Enable `AutoScaling` in the `features` option:

   ```
   features:
     - AutoScaling=true
   ```

   Enable the `Operator Webhook` feature:

   ```
   admissionWebhook:
     create: true
     mutation:
       pods: true
   ```

   For more information about `Operator Webhook`, see Enable Admission Controller in TiDB Operator.

2. Install or update TiDB Operator.

   To install or update TiDB Operator, see Deploy TiDB Operator in Kubernetes.

3. Confirm the resource configuration of the target TiDB cluster.

   Before using the auto-scaling feature on the target TiDB cluster, first you need to configure the CPU setting of the corresponding components. For example, you need to configure `spec.tikv.requests.cpu` in TiKV:

   ```
   spec:
     tikv:
       requests:
         cpu: "1"
   ```

```
  tidb:
    requests:
      cpu: "1"
```

6.5.2.2 TidbClusterAutoScaler

The TidbClusterAutoScaler CR object is used to control the behavior of the auto-scaling in the TiDB cluster. If you have used Horizontal Pod Autoscaler, presumably you are familiar with the notion TidbClusterAutoScaler. The following is an auto-scaling example in TiKV.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaling-demo
spec:
  cluster:
    name: auto-scaling-demo
    namespace: default
  monitor:
    name: auto-scaling-demo
    namespace: default
  tikv:
    minReplicas: 3
    maxReplicas: 4
    metrics:
      - type: "Resource"
        resource:
          name: "cpu"
          target:
            type: "Utilization"
            averageUtilization: 80
```

The TiDB component can be configured using spec.tidb. Currently, the auto-scaling API of TiDB is the same as that of TiKV.

In a TidbClusterAutoScaler object, the cluster attribute specifies the TiDB clusters to be auto-scaled. These clusters are marked by name and namespace. You need to provide the metrics collection and query service to TidbClusterAutoScaler because it captures resource usage through the metrics collection component. The monitor attribute refers to the TidbMonitor object. For more information, see Deploy Monitoring and Alerts for a TiDB Cluster.

For the external Prometheus other than TidbMonitor, you can fill in the Host by configuring spec.metricsUrl to specify the monitoring metrics collection service for the TiDB cluster. If you deploy the monitoring of the TiDB cluster using Helm, take the following

steps to specify `spec.metricsUrl`.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaling-demo
spec:
  cluster:
    name: auto-scaling-demo
    namespace: default
  metricsUrl: "http://${release_name}-prometheus.${namespace}.svc:9090"
  ......
```

6.5.2.3 Example

1. Run the following commands to quickly deploy a TiDB cluster with 3 PD instances, 3 TiKV instances, 2 TiDB instances, and the monitoring and the auto-scaling features.

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/examples/auto-scale/tidb-cluster.yaml -n ${
    ↪ namespace}
```

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/examples/auto-scale/tidb-monitor.yaml -n ${
    ↪ namespace}
```

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.1.15/examples/auto-scale/tidb-cluster-auto-scaler.
    ↪ yaml -n ${namespace}
```

2. After the TiDB cluster is created, expose the TiDB cluster service to the local machine by running the following command:

```
kubectl port-forward svc/auto-scaling-demo-tidb 4000:4000 &
```

Copy the following content and paste it to the local `sysbench.config` file:

```
mysql-host=127.0.0.1
mysql-port=4000
mysql-user=root
mysql-password=
mysql-db=test
time=120
threads=20
report-interval=5
db-driver=mysql
```

3. Prepare data and perform the stress test against the auto-scaling feature using sysbench.

   Copy the following content and paste it to the local `sysbench.config` file:

```
mysql-host=127.0.0.1
mysql-port=4000
mysql-user=root
mysql-password=
mysql-db=test
time=120
threads=20
report-interval=5
db-driver=mysql
```

   Prepare data by running the following command:

```
sysbench --config-file=${path-to-file}/sysbench.config
   ↪ oltp_point_select --tables=1 --table-size=20000 prepare
```

   Start the stress test:

```
sysbench --config-file=${path-to-file}/sysbench.config
   ↪ oltp_point_select --tables=1 --table-size=20000 run
```

   The command above will return the following result:

```
Initializing worker threads...

Threads started!

[ 5s ] thds: 20 tps: 37686.35 qps: 37686.35 (r/w/o: 37686.35/0.00/0.00)
   ↪  lat (ms,95%): 0.99 err/s: 0.00 reconn/s: 0.00
[ 10s ] thds: 20 tps: 38487.20 qps: 38487.20 (r/w/o:
   ↪ 38487.20/0.00/0.00) lat (ms,95%): 0.95 err/s: 0.00 reconn/s: 0.00
```

4. Create a new terminal session and view the Pod changing status of the TiDB cluster by running the following command:

```
watch -n1 "kubectl -n ${namespace} get pod"
```

   The output is as follows:

```
auto-scaling-demo-discovery-fbd95b679-f4cb9 1/1 Running 0         17m
auto-scaling-demo-monitor-6857c58564-ftkp4 3/3 Running  0         17m
auto-scaling-demo-pd-0                      1/1    Running  0         17m
auto-scaling-demo-tidb-0                    2/2    Running  0         15m
auto-scaling-demo-tidb-1                    2/2    Running  0         15m
auto-scaling-demo-tikv-0                    1/1    Running  0         15m
```

```
auto-scaling-demo-tikv-1                  1/1     Running   0          15m
auto-scaling-demo-tikv-2                  1/1     Running   0          15m
```

View the changing status of Pods and the TPS and QPS of sysbench. When new Pods are created in TiKV and TiDB, the TPS and QPS of sysbench increase significantly.

After sysbench finishes the test, the newly created Pods in TiKV and TiDB disappear automatically.

5. Destroy the environment by running the following commands:

```
kubectl delete tidbcluster auto-scaling-demo -n ${namespace}
kubectl delete tidbmonitor auto-scaling-demo -n ${namespace}
kubectl delete tidbclusterautoscaler auto-scaling-demo -n ${namespace}
```

6.5.2.4 `TidbClusterAutoScaler` configurations

1. Set the auto-scaling interval.

   Compared with the stateless web service, a distributed database software is often sensitive to the instance auto-scaling. You need to make sure that there is a certain interval between each auto-scaling in case scaling operations are too frequent.

   You can set the interval (in seconds) between each auto-scaling by configuring `spec.tikv.scaleInIntervalSeconds` and `spec.tikv.ScaleOutIntervalSeconds` in TiKV. This also applies to TiDB.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaler
spec:
  tidb:
    scaleInIntervalSeconds: 500
    ScaleOutIntervalSeconds: 300
  tikv:
    scaleInIntervalSeconds: 500
    ScaleOutIntervalSeconds: 300
```

2. Set the maximum value and the minimum value.

   You can set the maximum value and the minimum value of each component in `TidbClusterAutoScaler` to control the scaling range of `TiDB` and `TiKV`, which is similar to Horizontal Pod Autoscaler.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
```

```
  name: auto-scaling-demo
spec:
  tikv:
    minReplicas: 3
    maxReplicas: 4
  tidb:
    minReplicas: 2
    maxReplicas: 3
```

3. Set the CPU auto-scaling configurations.

   Currently, `TidbClusterAutoScaler` only supports CPU utilization based auto-scaling. The descriptive API is as follows. `averageUtilization` refers to the threshold of CPU utilization. If the utilization exceeds 80%, the auto-scaling is triggered.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: auto-scaling-demo
spec:
  tikv:
    minReplicas: 3
    maxReplicas: 4
    metrics:
      - type: "Resource"
        resource:
          name: "cpu"
          target:
            type: "Utilization"
            averageUtilization: 80
```

4. Set the time window configurations.

   The CPU utilization based auto-scaling allows `TidbClusterAutoScaler` to get the CPU metrics of `TiDB` and `TiKV` from the specified monitoring system. You can specify the time window of metrics collection.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbClusterAutoScaler
metadata:
  name: basic
  tidb:
    metricsTimeDuration: "1m"
    metrics:
      - type: "Resource"
        resource:
          name: "cpu"
```

```
    target:
      type: "Utilization"
      averageUtilization: 60
```

6.6 Backup and Restore

6.6.1 Backup and Restore Overview

This document describes how to perform backup and restore on the TiDB cluster in Kubernetes. The backup and restore tools used are BR, Dumpling, and TiDB Lightning.

TiDB Operator 1.1 and later versions implement the backup and restore methods using Custom Resource Definition (CRD):

- If your TiDB cluster version is v3.1 or later, refer to the following documents:
  - Back up Data to S3-Compatible Storage Using BR
  - Back up Data to GCS Using BR
  - Back up Data to PV Using BR
  - Restore Data from S3-Compatible Storage Using BR
  - Restore Data from GCS Using BR
  - Restore Data from PV Using BR
- If your TiDB cluster version is earlier than v3.1, refer to the following documents:
  - Back up Data to S3-Compatible Storage Using Dumpling
  - Back up Data to GCS Using Dumpling
  - Restore Data from S3-Compatible Storage Using TiDB Lightning
  - Restore Data from GCS Using TiDB Lightning

6.6.1.1 User scenarios

Dumpling is a data export tool that exports data stored in TiDB/MySQL as SQL or CSV data files to get the logic full backup or export. If you need to back up SST files (Key-Value pairs) directly or perform latency-insensitive incremental backup, refer to BR. For real-time incremental backup, refer to TiCDC.

TiDB Lightning is a tool used for fast full data import into a TiDB cluster. TiDB Lightning supports Dumpling or CSV format data source. You can use TiDB Lightning for the following two purposes:

- Quickly import large amounts of data
- Restore all backup data

BR is a command-line tool for distributed backup and restore of the TiDB cluster data. Compared with Dumpling and mydumper, BR is more suitable for scenarios of huge data volume. BR only supports TiDB v3.1 and later versions.

6.6.1.2 Backup CR fields

To back up data for a TiDB cluster in Kubernetes, you can create a `Backup` Custom Resource (CR) object. For detailed backup process, refer to documents listed in Backup and Restore Overview.

This section introduces the fields in the `Backup` CR.

6.6.1.2.1 General fields

- `.spec.metadata.namespace`: The namespace where the `Backup` CR is located.
- `.spec.toolImage`: The tool image used by `Backup`.
  - When using BR for backup, you can specify the BR version in this field.
    - If the field is not specified or the value is empty, the `pingcap/br:${tikv_version}` image is used for backup by default.
    - If the BR version is specified in this field, such as `.spec.toolImage: pingcap/br:v5.0.6`, the image of the specified version is used for backup.
    - If the BR version is not specified in the field, such as `.spec.toolImage : private/registry/br`, the `private/registry/br:${tikv_version}` image is used for backup.
  - When using Dumpling for backup, you can specify the Dumpling version in this field. For example, `spec.toolImage: pingcap/dumpling:v5.0.6`. If not specified, the Dumpling version specified in `TOOLKIT_V40` of the Backup Manager Dockerfile is used for backup by default.
  - TiDB Operator supports this configuration starting from v1.1.9.
- `.spec.tikvGCLifeTime`: The temporary `tikv_gc_life_time` time setting during the backup, which defaults to 72h.

  Before the backup begins, if the `tikv_gc_life_time` setting in the TiDB cluster is smaller than `spec.tikvGCLifeTime` set by the user, TiDB Operator adjusts the value of `tikv_gc_life_time` to the value of `spec.tikvGCLifeTime`. This operation makes sure that the backup data is not garbage-collected by TiKV.

  After the backup, no matter the backup is successful or not, as long as the previous `tikv_gc_life_time` value is smaller than `.spec.tikvGCLifeTime`, TiDB Operator will try to set `tikv_gc_life_time` to the previous value.

  In extreme cases, if TiDB Operator fails to access the database, TiDB Operator cannot automatically recover the value of `tikv_gc_life_time` and treats the backup as failed. At this time, you can view `tikv_gc_life_time` of the current TiDB cluster using the following statement:

```
select VARIABLE_NAME, VARIABLE_VALUE from mysql.tidb where
    ↪ VARIABLE_NAME like "tikv_gc_life_time";
```

In the output of the command above, if the value of `tikv_gc_life_time` is still larger than expected (usually 10m), you need to set `tikv_gc_life_time` back to the previous value manually:

- `.spec.cleanPolicy`: The cleaning policy for the backup data when the backup CR is deleted.

  Three clean policies are supported:

  - `Retain`: Under any circumstances, retain the backup data when deleting the backup CR.
  - `Delete`: Under any circumstances, delete the backup data when deleting the backup CR.
  - `OnFailure`: If the backup fails, delete the backup data when deleting the backup CR.

  If this field is not configured, or if you configure a value other than the three policies above, the backup data is retained.

  Note that in v1.1.2 and earlier versions, this field does not exist. The backup data is deleted along with the CR by default. For v1.1.3 or later versions, if you want to keep this earlier behavior, set this field to `Delete`.

- `.spec.from.host`: The address of the TiDB cluster to be backed up, which is the service name of the TiDB cluster to be exported, such as `basic-tidb`.

- `.spec.from.port`: The port of the TiDB cluster to be backed up.

- `.spec.from.user`: The accessing user of the TiDB cluster to be backed up.

- `.spec.from.secretName`: The secret that contains the password of the `.spec.from.user`.

- `.spec.from.tlsClientSecretName`: The secret of the certificate used during the backup.

  If TLS is enabled for the TiDB cluster, but you do not want to back up data using the `${cluster_name}-cluster-client-secret` as described in Enable TLS between TiDB Components, you can use the `.spec.from.tlsClient.tlsSecret` parameter to specify a secret for the backup. To generate the secret, run the following command:

```
kubectl create secret generic ${secret_name} --namespace=${namespace}
    ↪ --from-file=tls.crt=${cert_path} --from-file=tls.key=${key_path}
    ↪ --from-file=ca.crt=${ca_path}
```

- `.spec.storageClassName`: The persistent volume (PV) type specified for the backup operation.

- `.spec.storageSize`: The PV size specified for the backup operation (`100 Gi` by default). This value must be greater than the size of the TiDB cluster to be backed up.

  The PVC name corresponding to the `Backup` CR of a TiDB cluster is fixed. If the PVC already exists in the cluster namespace and the size is smaller than `spec.storageSize`, you need to delete this PVC and then run the Backup job.

- `.spec.tableFilter`: Specifies tables that match the table filter rules for BR or Dumpling. This field can be ignored by default.

  If the field is not configured, the default value of `tableFilter` is as follows:

```
tableFilter:
- "*.*"
- "!/^(mysql|test|INFORMATION_SCHEMA|PERFORMANCE_SCHEMA|METRICS_SCHEMA|
    ↪ INSPECTION_SCHEMA)$/.*"
```

  If you use BR to perform backup, BR backs up all schemas except the system schema.

> **Note:**
>
> To use the table filter to exclude `db.table`, you need to first add the `*.*` rule to include all tables. For example:
>
> ```
> tableFilter:
> - "*.*"
> - "!db.table"
> ```

6.6.1.2.2 BR fields

- `.spec.br.cluster`: The name of the cluster to be backed up.
- `.spec.br.clusterNamespace`: The `namespace` of the cluster to be backed up.
- `.spec.br.logLevel`: The log level (`info` by default).
- `.spec.br.statusAddr`: The listening address through which BR provides statistics. If not specified, BR does not listen on any status address by default.
- `.spec.br.concurrency`: The number of threads used by each TiKV process during backup. Defaults to `4` for backup and `128` for restore.
- `.spec.br.rateLimit`: The speed limit, in MB/s. If set to `4`, the speed limit is 4 MB/s. The speed limit is not set by default.
- `.spec.br.checksum`: Whether to verify the files after the backup is completed. Defaults to `true`.
- `.spec.br.timeAgo`: Backs up the data before `timeAgo`. If the parameter value is not specified (empty by default), it means backing up the current data. It supports data formats such as "1.5h" and "2h45m". See ParseDuration for more information.

- `.spec.br.sendCredToTikv`: Whether the BR process passes its AWS or GCP privileges to the TiKV process. Defaults to `true`.
- `.spec.br.options`: The extra arguments that BR supports. This field is supported since TiDB Operator v1.1.6. It accepts an array of strings and can be used to specify the last backup timestamp `--lastbackupts` for incremental backup.

6.6.1.2.3 S3 storage fields

- `.spec.s3.provider`: The supported S3-compatible storage provider. Options are as follows:
  - `alibaba`: Alibaba Cloud Object Storage System (OSS), formerly Aliyun
  - `digitalocean`: Digital Ocean Spaces
  - `dreamhost`: Dreamhost DreamObjects
  - `ibmcos`: IBM COS S3
  - `minio`: Minio Object Storage
  - `netease`: Netease Object Storage (NOS)
  - `wasabi`: Wasabi Object Storage
  - `other`: Any other S3 compatible provider
- `spec.s3.region`: If you want to use Amazon S3 for backup storage, configure this field as the region where Amazon S3 is located.
- `.spec.s3.bucket`: The name of the bucket compatible with S3 storage.
- `.spec.s3.prefix`: If you set this field, the value is used to make up the remote storage path `s3://${.spec.s3.bucket}/${.spec.s3.prefix}/backupName`.
- `.spec.s3.acl`: The supported access-control list (ACL) policies.

  Amazon S3 supports the following ACL options:
  - `private`
  - `public-read`
  - `public-read-write`
  - `authenticated-read`
  - `bucket-owner-read`
  - `bucket-owner-full-control`

  If the field is not configured, the policy defaults to `private`. For more information on the ACL policies, refer to AWS documentation.
- `.spec.s3.storageClass`: The supported storage class.

  Amazon S3 supports the following storage class options:
  - `STANDARD`

  - REDUCED_REDUNDANCY
  - STANDARD_IA
  - ONEZONE_IA
  - GLACIER
  - DEEP_ARCHIVE

  If the field is not configured, the storage class defaults to STANDARD_IA. For more information on storage classes, refer to AWS documentation.

6.6.1.2.4 GCS fields

- .spec.gcs.projectId: The unique identifier of the user project on GCP. To obtain the project ID, refer to GCP documentation.
- .spec.gcs.bucket: The name of the bucket which stores data.
- .spec.gcs.prefix: If you set this field, the value is used to make up the path of the remote storage: gcs://${.spec.gcs.bucket}/${.spec.gcs.prefix}/backupName. This field can be ignored.
- spec.gcs.storageClass: The supported storage class.

  GCS supports the following storage class options:

  - MULTI_REGIONAL
  - REGIONAL
  - NEARLINE
  - COLDLINE
  - DURABLE_REDUCED_AVAILABILITY

  If the field is not configured, the storage class defaults to COLDLINE. For more information on storage classes, refer to GCS documentation.

- .spec.gcs.objectAcl: The supported object access-control list (ACL) policies.

  GCS supports the following object ACL options:

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

  If the field is not configured, the policy defaults to private. For more information on the ACL policies, refer to GCS documentation.

- `.spec.gcs.bucketAcl`: The supported bucket access-control list (ACL) policies.

  GCS supports the following bucket ACL options:

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

  If the field is not configured, the policy defaults to `private`. For more information on the ACL policies, refer to GCS documentation.

6.6.1.2.5 Local storage fields

- `.spec.local.prefix`: The storage directory of the persistent volumes. If you set this field, the value is used to make up the storage path of the persistent volume: `local://${.spec.local.volumeMount.mountPath}/${.spec.local.prefix}/`.
- `.spec.local.volume`: The persistent volume configuration.
- `.spec.local.volumeMount`: The persistent volume mount configuration.

6.6.1.3 Restore CR fields

To restore data to a TiDB cluster in Kubernetes, you can create a `Restore` CR object. For detailed restore process, refer to documents listed in Backup and Restore Overview.

This section introduces the fields in the `Restore` CR.

- `.spec.metadata.namespace`: The namespace where the `Restore` CR is located.
- `.spec.toolImage:` The tools image used by `Restore`.

  - When using BR for restoring, you can specify the BR version in this field. For example,`spec.toolImage: pingcap/br:v4.0.10`. If not specified, `pingcap/br:${tikv_version}` is used for restoring by default.
  - When using Lightning for restoring, you can specify the Lightning version in this field. For example, `spec.toolImage: pingcap/lightning:v4.0.10`. If not specified, the Lightning version specified in `TOOLKIT_V40` of the Backup Manager Dockerfile is used for restoring by default.
  - TiDB Operator supports this configuration starting from v1.1.9.

- `.spec.to.host`: The address of the TiDB cluster to be restored.
- `.spec.to.port`: The port of the TiDB cluster to be restored.
- `.spec.to.user`: The accessing user of the TiDB cluster to be restored.

- `.spec.to.secretName`: The secret that contains the password of the `.spec.to.user`.
- `.spec.to.tlsClientSecretName`: The secret of the certificate used during the restore.

  If TLS is enabled for the TiDB cluster, but you do not want to restore data using the `${cluster_name}-cluster-client-secret` as described in Enable TLS between TiDB Components, you can use the `.spec.to.tlsClient.tlsSecret` parameter to specify a secret for the restore. To generate the secret, run the following command:

```
kubectl create secret generic ${secret_name} --namespace=${namespace}
    ↪ --from-file=tls.crt=${cert_path} --from-file=tls.key=${key_path}
    ↪ --from-file=ca.crt=${ca_path}
```

- `.spec.storageClassName`: The persistent volume (PV) type specified for the restore operation.
- `.spec.storageSize`: The PV size specified for the restore operation. This value must be greater than the size of the backup data.
- `.spec.tableFilter`: Specifies tables that match the table filter rules for BR. This field can be ignored by default.

  If the field is not configured, the default `tableFilter` value for TiDB Lightning is as follows:

```
tableFilter:
- "*.*"
- "!/^(mysql|test|INFORMATION_SCHEMA|PERFORMANCE_SCHEMA|METRICS_SCHEMA|
    ↪ INSPECTION_SCHEMA)$/.*"
```

  If this field is not configured, BR restores all the schemas in the backup file.

> **Note:**
>
> To use the table filter to exclude `db.table`, you need to first add the `*.*` rule to include all tables. For example:
>
> ```
> tableFilter:
> - "*.*"
> - "!db.table"
> ```

- `.spec.br`: BR-related configuration. Refer to BR fields.
- `.spec.s3`: S3-related configuration. Refer to S3 storage fields.
- `.spec.gcs`: GCS-related configuration. Refer to GCS fields.
- `.spec.local`: Persistent volume-related configuration. Refer to Local storage fields.

6.6.1.4 BackupSchedule CR fields

The `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`. `backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR.

The unique configuration items of `backupSchedule` are as follows:

- `.spec.maxBackups`: A backup retention policy, which determines the maximum number of backup files to be retained. When the number of backup files exceeds this value, the outdated backup file will be deleted. If you set this field to `0`, all backup items are retained.
- `.spec.maxReservedTime`: A backup retention policy based on time. For example, if you set the value of this field to `24h`, only backup files within the recent 24 hours are retained. All backup files older than this value are deleted. For the time format, refer to `func ParseDuration`. If you have set `.spec.maxBackups` and `.spec.maxReservedTime` at the same time, the latter takes effect.
- `.spec.schedule`: The time scheduling format of Cron. Refer to Cron for details.
- `.spec.pause`: `false` by default. If this field is set to `true`, the scheduled scheduling is paused. In this situation, the backup operation will not be performed even if the scheduling time is reached. During this pause, the backup Garbage Collection runs normally. If you change `true` to `false`, the scheduled full backup process is restarted.

6.6.1.5 Delete the Backup CR

You can delete the `Backup` CR or `BackupSchedule` CR by running the following commands:

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

If you use TiDB Operator v1.1.2 or an earlier version, or if you use TiDB Operator v1.1.3 or a later version and set the value of `spec.cleanPolicy` to `Delete`, TiDB Operator deletes the backup data when it deletes the CR.

In such cases, if you need to delete the namespace, it is recommended that you first delete all the `Backup`/`BackupSchedule` CRs and then delete the namespace.

If you delete the namespace before you delete the `Backup`/`BackupSchedule` CR, TiDB Operator will keep creating jobs to clean the backup data. However, because the namespace is in `Terminating` state, TiDB Operator fails to create such a job, which causes the namespace to be stuck in this state.

To address this issue, delete `finalizers` by running the following command:

```
kubectl edit backup ${name} -n ${namespace}
```

After deleting the `metadata.finalizers` configuration, you can delete the CR normally.

6.6.2 Grant Permissions to Remote Storage

This document describes how to grant permissions to access remote storage for backup and restore. During the backup process, TiDB cluster data is backed up to the remote storage. During the restore process, the backup data is restored from the remote storage to the TiDB cluster.

6.6.2.1 AWS account permissions

Amazon Web Service (AWS) provides different methods to grant permissions for different types of Kubernetes clusters. This document describes the following three methods.

6.6.2.1.1 Grant permissions by AccessKey and SecretKey

The AWS client can read `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` from the process environment variables to obtain the associated user or role permissions.

Create the `s3-secret` secret by running the following command. Use the AWS account's AccessKey and SecretKey. The secret stores the credential used for accessing S3-compatible storage.

```
kubectl create secret generic s3-secret --from-literal=access_key=xxx --from
    ↪ -literal=secret_key=yyy --namespace=test1
```

6.6.2.1.2 Grant permissions by associating IAM with Pod

If you associate the user's IAM role with the resources of the running Pods, the processes running in the Pods can have the permissions of the role. This method is provided by `kube2iam`.

> **Note:**
>
> - When you use this method to grant permissions, you can create the `kube2iam` environment in the Kubernetes cluster and deploy TiDB Operator and the TiDB cluster.
> - This method is not applicable to the `hostNetwork` mode. Make sure the value of `spec.tikv.hostNetwork` is set to `false`.

1. Create an IAM role.

   First, create an IAM User for your account.

   Then, Give the required permission to the IAM role you have created. Refer to Adding and Removing IAM Identity Permissions for details.

Because the `Backup` CR needs to access the Amazon S3 storage, the IAM role is granted the `AmazonS3FullAccess` permission.

2. Associate IAM with the TiKV Pod:

   When you use BR to back up TiDB data, the TiKV Pod also needs to perform read and write operations on S3-compatible storage as the BR Pod does. Therefore, you need to add annotations to the TiKV Pod to associate it with the IAM role.

```
kubectl edit tc demo1 -n test1
```

   Find `spec.tikv.annotations`, add this annotation to it: `iam.amazonaws.com/role : arn:aws:iam::123456789012:role/user`, and exit the editor. After the TiKV Pod is restarted, check whether the Pod has the annotation.

> **Note:**
>
> `arn:aws:iam::123456789012:role/user` is the IAM role created in Step 1.

6.6.2.1.3 Grant permissions by associating IAM with ServiceAccount

If you associate the user's IAM role with `serviceAccount` of Kubernetes, the Pods using the `serviceAccount` can have the permissions of the role. This method is provided by `EKS Pod Identity Webhook`.

When you use this method to grant permissions, you can create the EKS cluster and deploy TiDB Operator and the TiDB cluster.

1. Enable the IAM role for the `serviceAccount` in the cluster:

   Refer to AWS documentation.

2. Create the IAM role:

   Create an IAM role and grant the `AmazonS3FullAccess` permissions to the role. Edit the role's `Trust relationships`.

3. Associate IAM with the `ServiceAccount` resources.

```
kubectl annotate sa tidb-backup-manager eks.amazonaws.com/role-arn=arn:
    ↪ aws:iam::123456789012:role/user --namespace=test1
```

4. Associate the `ServiceAccount` with the TiKV Pod:

```
kubectl edit tc demo1 -n test1
```

Modify the value of `spec.tikv.serviceAccount` to `tidb-backup-manager`. After the TiKV Pod is restarted, check whether the Pod's `serviceAccountName` is changed.

Note:

`arn:aws:iam::123456789012:role/user` is the IAM role created in Step 2.

6.6.2.2 GCS account permissions

6.6.2.2.1 Grant permissions by the service account

Create the `gcs-secret` secret which stores the credential used to access GCS. The `google-credentials.json` file stores the service account key that you have downloaded from the GCP console. Refer to GCP documentation for details.

```
kubectl create secret generic gcs-secret --from-file=credentials=./google-
    ↪ credentials.json -n test1
```

6.6.3 Backup and Restore with S3-Compatible Storage

6.6.3.1 Back up Data to S3-Compatible Storage Using BR

This document describes how to back up the data of a TiDB cluster in AWS Kubernetes to the AWS storage using Helm charts. BR is used to get the logic backup of the TiDB cluster, and then this backup data is sent to the AWS storage.

The backup method described in this document is implemented using Custom Resource Definition (CRD) in TiDB Operator v1.1 or later versions.

6.6.3.1.1 Ad-hoc backup

Ad-hoc backup supports both full backup and incremental backup. It describes the backup by creating a `Backup` Custom Resource (CR) object. TiDB Operator performs the specific backup operation based on this `Backup` object. If an error occurs during the backup process, TiDB Operator does not retry, and you need to handle this error manually.

To better describe the backup process, this document provides examples in which the data of the `demo1` TiDB cluster in the `test1` Kubernetes namespace is backed up to AWS storage.

Prerequisites for ad-hoc backup

Before you perform ad-hoc backup, AWS account permissions need to be granted. This section describes three methods to grant AWS account permissions.

Note:

If TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR will automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.tikvGCLifeTime` and `spec.from` fields in the `Backup` CR. In addition, you can skip the steps of creating the `backup-demo1-tidb-secret` secret and configuring database account privileges.

1. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `test1` namespace:

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

2. Grant permissions to the remote storage.

   To grant permissions to access the S3-compatible remote storage, refer to AWS account permissions.

   If you use Ceph as the backend storage for testing, you can grant permissions by using AccessKey and SecretKey.

3. Create the `backup-demo1-tidb-secret` secret which stores the account and password needed to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=${password} --namespace=test1
```

Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the backup, the `Backup` CR needs a database account with these privileges to adjust the GC time.

Process of ad-hoc backup

- If you grant permissions by importing AccessKey and SecretKey, create the `Backup` CR, and back up cluster data as described below:

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

The content of `backup-aws-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  br:
    cluster: demo1
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
    # options:
    # - --lastbackupts=420134118382108673
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

- If you grant permissions by associating IAM with Pod, create the `Backup` CR, and back up cluster data as described below:

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

The content of `backup-aws-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
```

```
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  backupType: full
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # options:
    # - --lastbackupts=420134118382108673
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

- If you grant permissions by associating IAM with ServiceAccount, create the `Backup` CR, and back up cluster data as described below:

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

The content of `backup-aws-s3.yaml` is as follows:

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

```
  cluster: demo1
  sendCredToTikv: false
  clusterNamespace: test1
  # logLevel: info
  # statusAddr: ${status_addr}
  # concurrency: 4
  # rateLimit: 0
  # timeAgo: ${time}
  # checksum: true
  # options:
  # - --lastbackupts=420134118382108673
# Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
from:
  host: ${tidb_host}
  port: ${tidb_port}
  user: ${tidb_user}
  secretName: backup-demo1-tidb-secret
s3:
  provider: aws
  region: us-west-1
  bucket: my-bucket
  prefix: my-folder
```

The three examples above use three methods to grant permissions to back up data to Amazon S3 storage. The `acl`, `endpoint`, `storageClass` configuration items of Amazon S3 can be ignored. For more information about S3-compatible storage configuration, refer to S3 storage fields.

In the examples above, some parameters in `.spec.br` can be ignored, such as `logLevel` ↪ , `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.

Since TiDB Operator v1.1.6, if you want to back up data incrementally, you only need to specify the last backup timestamp `--lastbackupts` in `spec.br.options`. For the limitations of incremental backup, refer to Use BR to Back up and Restore Data.

For more information about the `Backup` CR fields, refer to Backup CR fields.

After you create the `Backup` CR, view the backup status by running the following command:

```
shell kubectl get bk -n test1 -o wide
```

Backup CR examples

Back up data of all clusters

```
---
apiVersion: pingcap.com/v1alpha1
```

```
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

Back up data of a single database

The following example backs up data of the `db1` database.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
```

```
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

Back up data of a single table

The following example backs up data of the `db1.table1` table.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

Back up data of multiple tables using the table filter

The following example backs up data of the `db1.table1` table and `db1.table2` table.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
```

```
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  # ...
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

6.6.3.1.2 Scheduled full backup

You can set a backup policy to perform scheduled backups of the TiDB cluster, and set a backup retention policy to avoid excessive backup items. A scheduled full backup is described by a custom `BackupSchedule` CR object. A full backup is triggered at each backup time point. Its underlying implementation is the ad-hoc full backup.

Prerequisites for scheduled full backup

The prerequisites for the scheduled full backup is the same as the prerequisites for ad-hoc backup.

Process of scheduled full backup

- If you grant permissions by importing AccessKey and SecretKey, create the `BackupSchedule` CR, and back up cluster data as described below:

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

The content of `backup-scheduler-aws-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
      # sendCredToTikv: true
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

- If you grant permissions by associating IAM with the Pod, create the `BackupSchedule` CR, and back up cluster data as described below:

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

The content of `backup-scheduler-aws-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    # Clean outdated backup data based on maxBackups or maxReservedTime
       ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: false
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

- If you grant permissions by associating IAM with ServiceAccount, create the `BackupSchedule` CR, and back up cluster data as described below:

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

The content of `backup-scheduler-aws-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    serviceAccount: tidb-backup-manager
    # Clean outdated backup data based on maxBackups or maxReservedTime
       ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: false
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

After creating the scheduled full backup, use the following command to check the backup status:

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

During cluster recovery, you need to specify the backup path. You can use the following command to check all the backup items. The names of these backups are prefixed with the scheduled snapshot backup name:

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

From the example above, you can see that the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`.

`backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR. For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.

6.6.3.1.3 Delete the backup CR

Refer to Delete the Backup CR.

6.6.3.1.4 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

6.6.3.2 Restore Data from S3-Compatible Storage Using BR

This document describes how to restore the TiDB cluster data backed up using TiDB Operator in Kubernetes. BR is used to perform the restore.

The restore method described in this document is implemented based on Custom Resource Definition (CRD) in TiDB Operator v1.1 or later versions.

This document shows an example in which the backup data stored in the specified path on the Amazon S3 storage is restored to the TiDB cluster.

6.6.3.2.1 Prerequisites

> **Note:**
>
> If TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR will automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.to` fields in the `Restore` CR. In addition, you can skip the steps of creating the `restore -demo2-tidb-secret` secret and configuring database account privileges.

1. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `test2` namespace:

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. Grant permissions to the remote storage.

   To grant permissions to access S3-compatible remote storage, refer to AWS account permissions.

   If you use Ceph as the backend storage for testing, you can grant permissions by using AccessKey and SecretKey.

3. Create the `restore-demo2-tidb-secret` secret which stores the account and password needed to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ password=${password} --namespace=test2
```

6.6.3.2.2 Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the restore, the `Restore` CR needs a database account with these privileges to adjust the GC time.

6.6.3.2.3 Restore process

- If you grant permissions by importing AccessKey and SecretKey, create the `Restore` CR, and restore cluster data as described below:

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

  The content of `restore-aws-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test2
spec:
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status_addr}
```

```
  # concurrency: 4
  # rateLimit: 0
  # timeAgo: ${time}
  # checksum: true
  # sendCredToTikv: true
# # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
# to:
#   host: ${tidb_host}
#   port: ${tidb_port}
#   user: ${tidb_user}
#   secretName: restore-demo2-tidb-secret
s3:
  provider: aws
  secretName: s3-secret
  region: us-west-1
  bucket: my-bucket
  prefix: my-folder
```

- If you grant permissions by associating IAM with Pod, create the `Restore` CR, and restore cluster data as described below:

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

The content of `restore-aws-s3.yaml` is as follows:

“‘yaml

---

apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
name: demo2-restore-s3
namespace: test2
annotations:
iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
br:
cluster: demo2
sendCredToTikv: false
clusterNamespace: test2
logLevel: info
statusAddr: ${status_addr}
concurrency: 4
rateLimit: 0
timeAgo: ${time}
checksum: true
Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8

```
“‘yaml
to:
  host: ${tidb_host}
  port: ${tidb_port}
  user: ${tidb_user}
  secretName: restore-demo2-tidb-secret
s3:
  provider: aws
  region: us-west-1
  bucket: my-bucket
  prefix: my-folder
“‘
```

- If you grant permissions by associating IAM with ServiceAccount, create the `Restore` CR, and restore cluster data as described below:

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

The content of `restore-aws-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test2
spec:
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo2
    sendCredToTikv: false
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
```

```
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-folder
```

After creating the `Restore` CR, execute the following command to check the restore status:

```
kubectl get rt -n test2 -o wide
```

The examples above restore data from the `spec.s3.prefix` folder of the `spec.s3.bucket` bucket on Amazon S3 storage to the `demo2` TiDB cluster in the `test2` namespace. For more information about S3-compatible storage configuration, refer to S3 storage fields.

In the examples above, some parameters in `.spec.br` can be ignored, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.

For more information about the `Restore` CR fields, refer to Restore CR fields.

6.6.3.2.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

6.6.3.3 Back up Data to S3-Compatible Storage Using Dumpling

This document describes how to back up the data of the TiDB cluster in Kubernetes to the S3-compatible storage. "Backup" in this document refers to full backup (ad-hoc full backup and scheduled full backup). For the underlying implementation, Dumpling is used to get the logic backup of the TiDB cluster, and then this backup data is sent to the S3-compatible storage.

The backup method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator v1.1 or later versions.

6.6.3.3.1 Ad-hoc full backup to S3-compatible storage

Ad-hoc full backup describes the backup by creating a `Backup` custom resource (CR) object. TiDB Operator performs the specific backup operation based on this `Backup` object. If an error occurs during the backup process, TiDB Operator does not retry and you need to handle this error manually.

For the current S3-compatible storage types, Ceph and Amazon S3 work normally as tested. Therefore, this document shows examples in which the data of the `demo1` TiDB cluster in the `tidb-cluster` Kubernetes namespace is backed up to Ceph and Amazon S3 respectively.

Prerequisites for ad-hoc full backup

1. Execute the following command to create the role-based access control (RBAC) resources in the `tidb-cluster` namespace based on backup-rbac.yaml:

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

2. Grant permissions to the remote storage.

   To grant permissions to access S3-compatible remote storage, refer to AWS account permissions.

   If you use Ceph as the backend storage for testing, you can grant permissions by using AccessKey and SecretKey.

3. Create the `backup-demo1-tidb-secret` secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=${password} --namespace=tidb-cluster
```

Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the backup, the `Backup` CR needs a database account with these privileges to adjust the GC time.
- The global privileges: `SELECT`, `RELOAD`, `LOCK TABLES` and `REPLICATION CLIENT`

An example for creating a backup user:

```
CREATE USER 'backup'@'%' IDENTIFIED BY '...';
GRANT
  SELECT, RELOAD, LOCK TABLES, REPLICATION CLIENT
  ON *.*
  TO 'backup'@'%';
GRANT
  UPDATE, SELECT
  ON mysql.tidb
  TO 'backup'@'%';
```

Ad-hoc backup process

Note:

Because of the `rclone` issue, if the backup data is stored in Amazon S3 and the `AWS-KMS` encryption is enabled, you need to add the following `spec.s3.options` configuration to the YAML file in the examples of this section:

```
spec:
  ...
  s3:
    ...
    options:
    - --ignore-checksum
```

Note:

This section lists multiple storage access methods. Only follow the method that matches your situation.

The methods are as follows:

- Amazon S3 by importing AccessKey and SecretKey
- Ceph by importing AccessKey and SecretKey
- Amazon S3 by binding IAM with Pod
- Amazon S3 by binding IAM with ServiceAccount

- Create the `Backup` CR, and back up cluster data to Amazon S3 by importing AccessKey and SecretKey to grant permissions:

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

The content of `backup-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: tidb-cluster
spec:
  from:
    host: ${tidb_host}
    port: ${tidb_port}
```

```
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: aws
    secretName: s3-secret
    region: ${region}
    bucket: ${bucket}
    # prefix: ${prefix}
    # storageClass: STANDARD_IA
    # acl: private
    # endpoint:
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

- Create the `Backup` CR, and back up data to Ceph by importing AccessKey and SecretKey to grant permissions:

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

The content of `backup-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: tidb-cluster
spec:
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  s3:
    provider: ceph
    secretName: s3-secret
    endpoint: ${endpoint}
    # prefix: ${prefix}
    bucket: ${bucket}
# dumpling:
```

```
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

- Create the `Backup` CR, and back up data to Amazon S3 by binding IAM with Pod to grant permissions:

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

The content of `backup-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
name: demo1-backup-s3
namespace: tidb-cluster
annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
backupType: full
from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
s3:
    provider: aws
    region: ${region}
    bucket: ${bucket}
    # prefix: ${prefix}
    # storageClass: STANDARD_IA
    # acl: private
    # endpoint:
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

- Create the `Backup` CR, and back up data to Amazon S3 by binding IAM with ServiceAccount to grant permissions:

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

The content of `backup-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
name: demo1-backup-s3
namespace: tidb-cluster
spec:
backupType: full
serviceAccount: tidb-backup-manager
from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
s3:
    provider: aws
    region: ${region}
    bucket: ${bucket}
    # prefix: ${prefix}
    # storageClass: STANDARD_IA
    # acl: private
    # endpoint:
# dumpling:
#  options:
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  # storageClassName: local-storage
  storageSize: 10Gi
```

In the examples above, all data of the TiDB cluster is exported and backed up to Amazon S3 or Ceph. You can ignore the `acl`, `endpoint`, and `storageClass` fields in the Amazon S3 configuration. Other S3-compatible storages can also use a configuration similar to that of Amazon S3. You can also leave the fields empty if you do not need to configure them, as shown in the above Ceph configuration. For more information about S3-compatible storage configuration, refer to S3 storage fields.

`spec.dumpling` refers to Dumpling-related configuration. You can specify Dumpling's operation parameters in the `options` field. See Dumpling Option list for more information. These configuration items of Dumpling can be ignored by default. When these items are not specified, the default values of `options` fields are as follows:

```
options:
- --threads=16
- --rows=10000
```

For more information about the `Backup` CR fields, refer to Backup CR fields.

After creating the `Backup` CR, use the following command to check the backup status:

```
kubectl get bk -n tidb-cluster -owide
```

To get detailed information on a backup job, use the following command. For `$backup_job_name` in the command, use the name from the output of the previous command.

```
kubectl describe bk -n tidb-cluster $backup_job_name
```

To run ad-hoc backup again, you need to delete the backup CR and create it again.

6.6.3.3.2 Scheduled full backup to S3-compatible storage

You can set a backup policy to perform scheduled backups of the TiDB cluster, and set a backup retention policy to avoid excessive backup items. A scheduled full backup is described by a custom `BackupSchedule` CR object. A full backup is triggered at each backup time point. Its underlying implementation is the ad-hoc full backup.

Prerequisites for scheduled backup

The prerequisites for the scheduled backup is the same as the prerequisites for ad-hoc full backup.

Scheduled backup process

Note:

Because of the `rclone` issue, if the backup data is stored in Amazon S3 and the `AWS-KMS` encryption is enabled, you need to add the following `spec ↪ .backupTemplate.s3.options` configuration to the YAML file in the examples of this section:

```
spec:
  ...
  backupTemplate:
    ...
```

```
  s3:
    ...
    options:
    - --ignore-checksum
```

- Create the `BackupSchedule` CR to enable the scheduled full backup to Amazon S3 by importing AccessKey and SecretKey to grant permissions:

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

The content of `backup-schedule-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: tidb-cluster
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      secretName: s3-secret
      region: ${region}
      bucket: ${bucket}
      # prefix: ${prefix}
      # storageClass: STANDARD_IA
      # acl: private
      # endpoint:
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
```

```
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

- Create the `BackupSchedule` CR to enable the scheduled full backup to Ceph by importing AccessKey and SecretKey to grant permissions:

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

The content of `backup-schedule-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-ceph
  namespace: tidb-cluster
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: ceph
      secretName: s3-secret
      endpoint: ${endpoint}
      bucket: ${bucket}
      # prefix: ${prefix}
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

- Create the `BackupSchedule` CR to enable the scheduled full backup, and back up the cluster data to Amazon S3 by binding IAM with Pod to grant permissions:

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

The content of `backup-schedule-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: tidb-cluster
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: ${region}
      bucket: ${bucket}
      # prefix: ${prefix}
      # storageClass: STANDARD_IA
      # acl: private
      # endpoint:
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

- Create the `BackupSchedule` CR to enable the scheduled full backup, and back up the cluster data to Amazon S3 by binding IAM with ServiceAccount to grant permissions:

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

The content of `backup-schedule-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: tidb-cluster
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  serviceAccount: tidb-backup-manager
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    s3:
      provider: aws
      region: ${region}
      bucket: ${bucket}
      # prefix: ${prefix}
      # storageClass: STANDARD_IA
      # acl: private
      # endpoint:
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

After creating the scheduled full backup, you can use the following command to check the backup status:

```
kubectl get bks -n tidb-cluster -owide
```

You can use the following command to check all the backup items:

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

From the example above, you can see that the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`.

`backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR. For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.

> **Note:**
>
> TiDB Operator creates a PVC used for both ad-hoc full backup and scheduled full backup. The backup data is stored in PV first and then uploaded to remote storage. If you want to delete this PVC after the backup is completed, you can refer to Delete Resource to delete the backup Pod first, and then delete the PVC.
>
> If the backup data is successfully uploaded to remote storage, TiDB Operator automatically deletes the local data. If the upload fails, the local data is retained.

6.6.3.3.3 Delete the backup CR

Refer to Delete the Backup CR.

6.6.3.3.4 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

6.6.3.4 Restore Data from S3-Compatible Storage Using TiDB Lightning

This document describes how to restore the TiDB cluster data backed up using TiDB Operator in Kubernetes.

The restore method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator v1.1 or later versions. For the underlying implementation, TiDB Lightning TiDB-backend is used to perform the restore.

TiDB Lightning supports three backends: `Importer-backend`, `Local-backend`, and `TiDB-backend`. For the differences of these backends and how to choose backends, see TiDB Lightning Backends. To import data using `Importer-backend` or `Local-backend`, see Import Data.

6.6.3.4.1 Prerequisites

1. Download backup-rbac.yaml and execute the following command to create the role-based access control (RBAC) resources in the test2 namespace:

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. Grant permissions to the remote storage.

   To grant permissions to access S3-compatible remote storage, refer to AWS account permissions.

   If you use Ceph as the backend storage for testing, you can grant permissions by using AccessKey and SecretKey.

3. Create the restore-demo2-tidb-secret secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ password=${password} --namespace=test2
```

6.6.3.4.2 Required database account privileges

Privileges	Scope
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables

6.6.3.4.3 Restore process

Note:

Because of the rclone issue, if the backup data is stored in Amazon S3 and the AWS-KMS encryption is enabled, you need to add the following spec.s3.
↪ options configuration to the YAML file in the examples of this section:

```
spec:
  ...
  s3:
    ...
    options:
    - --ignore-checksum
```

- Create the `Restore` CR, and restore the cluster data from Ceph by importing AccessKey and SecretKey to grant permissions:

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

The content of `restore.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
spec:
  backupType: full
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: ceph
    endpoint: ${endpoint}
    secretName: s3-secret
    path: s3://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

- Create the `Restore` CR, and restore the cluster data from Amazon S3 by importing AccessKey and SecretKey to grant permissions:

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

The `restore.yaml` file has the following content:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
spec:
  backupType: full
  to:
```

```
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: ${region}
    secretName: s3-secret
    path: s3://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

- Create the `Restore` CR, and restore the cluster data from Amazon S3 by binding IAM with Pod to grant permissions:

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

The content of `restore.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  backupType: full
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: ${region}
    path: s3://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

- Create the `Restore` CR, and restore the cluster data from Amazon S3 by binding IAM with ServiceAccount to grant permissions:

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

The content of `restore.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
  s3:
    provider: aws
    region: ${region}
    path: s3://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

After creating the `Restore` CR, execute the following command to check the restore status:

```
kubectl get rt -n test2 -owide
```

The example above restores data from the `spec.s3.path` path on S3-compatible storage to the `spec.to.host` TiDB cluster. For more information about S3-compatible storage configuration, refer to S3 storage fields.

For more information about the `Restore` CR fields, refer to Restore CR fields.

Note:

TiDB Operator creates a PVC for data recovery. The backup data is downloaded from the remote storage to the PV first, and then restored. If you want to delete this PVC after the recovery is completed, you can refer to Delete Resource to delete the recovery Pod first, and then delete the PVC.

6.6.3.4.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

6.6.4 Backup and Restore with GCS

6.6.4.1 Back up Data to GCS Using BR

This document describes how to back up the data of a TiDB cluster in Kubernetes to Google Cloud Storage (GCS). BR is used to get the backup of the TiDB cluster, and then the backup data is sent to GCS.

The backup method described in this document is implemented using Custom Resource Definition (CRD) in TiDB Operator v1.1 or later versions.

6.6.4.1.1 Ad-hoc backup

Ad-hoc backup supports both full backup and incremental backup. It describes the backup by creating a `Backup` Custom Resource (CR) object. TiDB Operator performs the specific backup operation based on this `Backup` object. If an error occurs during the backup process, TiDB Operator does not retry, and you need to handle this error manually.

This document provides examples in which the data of the `demo1` TiDB cluster in the `test1` Kubernetes namespace is backed up to GCS.

Prerequisites for ad-hoc backup

> **Note:**
>
> If TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR will automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.tikvGCLifeTime` and `spec.from` fields in the `Backup` CR. In addition, you can skip the steps of creating the `backup-demo1-tidb-secret` secret and configuring database account privileges.

1. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `test1` namespace:

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

2. Grant permissions to the remote storage.

   Refer to GCS account permissions.

3. Create the `backup-demo1-tidb-secret` secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=<password> --namespace=test1
```

Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the backup, the `Backup` CR needs a database account with these privileges to adjust the GC time.

Process of ad-hoc backup

1. Create the `Backup` CR, and back up cluster data to GCS as described below:

```
kubectl apply -f backup-gcs.yaml
```

The content of `backup-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
    # sendCredToTikv: true
    # options:
    # - --lastbackupts=420134118382108673
  gcs:
    projectId: ${project_id}
```

```
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

In the example above, some parameters in `spec.br` can be ignored, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.

Since TiDB Operator v1.1.6, if you want to back up data incrementally, you only need to specify the last backup timestamp `--lastbackupts` in `spec.br.options`. For the limitations of incremental backup, refer to Use BR to Back up and Restore Data.

The example above backs up all data in the TiDB cluster to GCS. Some parameters in `spec.gcs` can be ignored, such as `location`, `objectAcl`, and `storageClass`. For more information about GCS configuration, refer to GCS fields.

For more information about the `Backup` CR fields, refer to Backup CR fields.

2. After creating the `Backup` CR, use the following command to check the backup status:

```
kubectl get bk -n test1 -owide
```

Backup CR examples

Back up data of all clusters

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
  gcs:
    projectId: ${project_id}
```

```
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

Back up data of a single database

The following example backs up data of the `db1` database.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    clusterNamespace: test1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

Back up data of a single table

The following example backs up data of the `db1.table1` table.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
```

```
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    clusterNamespace: test1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

Back up data of multiple tables using the table filter

The following example backs up data of the `db1.table1` table and `db1.table2` table.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  br:
```

```
    cluster: demo1
    clusterNamespace: test1
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

6.6.4.1.2 Scheduled full backup

You can set a backup policy to perform scheduled backups of the TiDB cluster, and set a backup retention policy to avoid excessive backup items. A scheduled full backup is described by a custom `BackupSchedule` CR object. A full backup is triggered at each backup time point. Its underlying implementation is the ad-hoc full backup.

Prerequisites for scheduled full backup

The prerequisites for the scheduled full backup is the same with the prerequisites for ad-hoc backup.

Process of scheduled full backup

1. Create the `BackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f backup-schedule-gcs.yaml
```

The content of `backup-schedule-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-gcs
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    # Clean outdated backup data based on maxBackups or maxReservedTime
       ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
```

```
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
    # sendCredToTikv: true
  gcs:
    secretName: gcs-secret
    projectId: ${project_id}
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

2. After creating the scheduled full backup, use the following command to check the backup status:

```
kubectl get bks -n test1 -owide
```

Use the following command to check all the backup items:

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

From the example above, you can see that the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`.

`backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR. For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.

6.6.4.1.3 Delete the backup CR

Refer to Delete the Backup CR.

6.6.4.1.4 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

6.6.4.2 Restore Data from GCS Using BR

This document describes how to restore the TiDB cluster data backed up using TiDB Operator in Kubernetes. BR is used to perform the restore.

The restore method described in this document is implemented based on Custom Resource Definition (CRD) in TiDB Operator v1.1 or later versions.

This document shows an example in which the backup data stored in the specified path on Google Cloud Storage (GCS) is restored to the TiDB cluster.

6.6.4.2.1 Prerequisites

> **Note:**
>
> If TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR will automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.to` fields in the `Restore` CR. In addition, you can skip the steps of creating the `restore -demo2-tidb-secret` secret and configuring database account privileges.

1. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `test2` namespace:

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. Grant permissions to the remote storage.

   Refer to GCS account permissions.

3. Create the `restore-demo2-tidb-secret` secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ user=root --from-literal=password=<password> --namespace=test2
```

6.6.4.2.2 Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the restore, the `Restore` CR needs a database account with these privileges to adjust the GC time.

6.6.4.2.3 Restore Process

1. Create the `Restore` custom resource (CR), and restore the specified data to your cluster:

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

The content of `restore.yaml` file is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-gcs
  namespace: test2
spec:
  # backupType: full
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
    # sendCredToTikv: true
  # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  # to:
  #   host: ${tidb_host}
  #   port: ${tidb_port}
  #   user: ${tidb_user}
  #   secretName: restore-demo2-tidb-secret
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: ${bucket}
    prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

2. After creating the `Restore` CR, execute the following command to check the restore status:

```
kubectl get rt -n test2 -owide
```

The example above restores data from the `spec.gcs.prefix` folder of the `spec.gcs.bucket` bucket on GCS to the `demo2` TiDB cluster in the `test2` namespace. For more information about GCS configuration, refer to GCS fields.

In the examples above, some parameters in `.spec.br` can be ignored, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.

For more information about the `Restore` CR fields, refer to Restore CR fields.

6.6.4.2.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

6.6.4.3 Back up Data to GCS Using Dumpling

This document describes how to back up the data of the TiDB cluster in Kubernetes to Google Cloud Storage (GCS). "Backup" in this document refers to full backup (ad-hoc full backup and scheduled full backup). Dumpling is used to get the logic backup of the TiDB cluster, and then this backup data is sent to the remote GCS.

The backup method described in this document is implemented using CustomResourceDefinition (CRD) in TiDB Operator v1.1 or later versions.

6.6.4.3.1 Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the backup, the `Backup` CR needs a database account with these privileges to adjust the GC time.
- SELECT
- RELOAD
- LOCK TABLES
- REPLICATION CLIENT

6.6.4.3.2 Ad-hoc full backup to GCS

Ad-hoc full backup describes a backup operation by creating a `Backup` custom resource (CR) object. TiDB Operator performs the specific backup operation based on this `Backup` object. If an error occurs during the backup process, TiDB Operator does not retry and you need to handle this error manually.

To better explain how to perform the backup operation, this document shows an example in which the data of the `demo1` TiDB cluster is backed up to the `test1` Kubernetes namespace.

Prerequisites for ad-hoc full backup

1. Download backup-rbac.yaml and execute the following command to create the role-based access control (RBAC) resources in the `test1` namespace:

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

2. Grant permissions to the remote storage.

   Refer to GCS account permissions.

3. Create the `backup-demo1-tidb-secret` secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=${password} --namespace=test1
```

Ad-hoc backup process

1. Create the `Backup` CR and back up data to GCS:

```
kubectl apply -f backup-gcs.yaml
```

   The content of `backup-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  from:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: backup-demo1-tidb-secret
  gcs:
    secretName: gcs-secret
    projectId: ${project_id}
    bucket: ${bucket}
    # prefix: ${prefix}
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
    # bucketAcl: private
# dumpling:
#  options:
```

```
#  - --threads=16
#  - --rows=10000
#  tableFilter:
#  - "test.*"
  storageClassName: local-storage
  storageSize: 10Gi
```

The example above backs up all data in the TiDB cluster to GCS. Some parameters in `spec.gcs` can be ignored, such as `location`, `objectAcl`, `bucketAcl`, and `storageClass`. For more information about GCS configuration, refer to GCS fields.

`spec.dumpling` refers to Dumpling-related configuration. You can specify Dumpling's operation parameters in the `options` field. See Dumpling Option list for more information. These configuration items of Dumpling can be ignored by default. When these items are not specified, the default values of `options` fields are as follows:

```
options:
- --threads=16
- --rows=10000
```

For more information about the `Backup` CR fields, refer to Backup CR fields.

2. After creating the `Backup` CR, use the following command to check the backup status:

```
kubectl get bk -n test1 -owide
```

6.6.4.3.3 Scheduled full backup to GCS

You can set a backup policy to perform scheduled backups of the TiDB cluster, and set a backup retention policy to avoid excessive backup items. A scheduled full backup is described by a custom `BackupSchedule` CR object. A full backup is triggered at each backup time point. Its underlying implementation is the ad-hoc full backup.

Prerequisites for scheduled backup

The prerequisites for the scheduled backup is the same as the prerequisites for ad-hoc full backup.

Scheduled backup process

1. Create the `BackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f backup-schedule-gcs.yaml
```

The content of `backup-schedule-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
```

```
metadata:
  name: demo1-backup-schedule-gcs
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    from:
      host: ${tidb_host}
      port: ${tidb_port}
      user: ${tidb_user}
      secretName: backup-demo1-tidb-secret
    gcs:
      secretName: gcs-secret
      projectId: ${project_id}
      bucket: ${bucket}
      # prefix: ${prefix}
      # location: us-east1
      # storageClass: STANDARD_IA
      # objectAcl: private
      # bucketAcl: private
  # dumpling:
  #  options:
  #  - --threads=16
  #  - --rows=10000
  #  tableFilter:
  #  - "test.*"
    # storageClassName: local-storage
    storageSize: 10Gi
```

2. After creating the scheduled full backup, use the following command to check the backup status:

```
kubectl get bks -n test1 -owide
```

Use the following command to check all the backup items:

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

From the example above, you can see that the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`.

`backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR. For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.

> **Note:**
>
> TiDB Operator creates a PVC used for both ad-hoc full backup and scheduled full backup. The backup data is stored in PV first and then uploaded to remote storage. If you want to delete this PVC after the backup is completed, you can refer to Delete Resource to delete the backup Pod first, and then delete the PVC.
>
> If the backup data is successfully uploaded to remote storage, TiDB Operator automatically deletes the local data. If the upload fails, the local data is retained.

6.6.4.3.4 Delete the backup CR

Refer to Delete the Backup CR.

6.6.4.3.5 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

6.6.4.4 Restore Data from GCS

This document describes how to restore the TiDB cluster data backed up using TiDB Operator in Kubernetes.

The restore method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator v1.1 or later versions. For the underlying implementation, TiDB Lightning TiDB-backend is used to perform the restore.

TiDB Lightning supports three backends: `Importer-backend`, `Local-backend`, and `TiDB-backend`. For the differences of these backends and how to choose backends, see TiDB Lightning Backends. To import data using `Importer-backend` or `Local-backend`, see Import Data.

This document shows an example in which the backup data stored in the specified path on Google Cloud Storage (GCS) is restored to the TiDB cluster.

6.6.4.4.1 Prerequisites

1. Download `backup-rbac.yaml` and execute the following command to create the role-based access control (RBAC) resources in the `test2` namespace:

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. Grant permissions to the remote storage.

   Refer to GCS account permissions.

3. Create the `restore-demo2-tidb-secret` secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ user=root --from-literal=password=${password} --namespace=test2
```

6.6.4.4.2 Required database account privileges

Privileges	Scope
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables

6.6.4.4.3 Restore process

1. Create the restore custom resource (CR) and restore the backup data to the TiDB cluster:

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

   The `restore.yaml` file has the following content:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore
  namespace: test2
spec:
  to:
    host: ${tidb_host}
    port: ${tidb_port}
    user: ${tidb_user}
    secretName: restore-demo2-tidb-secret
```

```
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    path: gcs://${backup_path}
  # storageClassName: local-storage
  storageSize: 1Gi
```

The example above restores data from the `spec.gcs.path` path on GCS to the `spec.to.host` TiDB cluster. For more information about GCS configuration, refer to GCS fields.

For more information about the `Restore` CR fields, refer to Restore CR fields.

2. After creating the `Restore` CR, execute the following command to check the restore status:

```
shell kubectl get rt -n test2 -owide
```

> **Note:**
>
> TiDB Operator creates a PVC for data recovery. The backup data is downloaded from the remote storage to the PV first, and then restored. If you want to delete this PVC after the recovery is completed, you can refer to Delete Resource to delete the recovery Pod first, and then delete the PVC.

6.6.4.4.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

6.6.5 Backup and Restore with Persistent Volumes

6.6.5.1 Back up Data to PV Using BR

This document describes how to back up the data of a TiDB cluster in Kubernetes to Persistent Volumes (PVs). BR is used to get the backup of the TiDB cluster, and then the backup data is sent to PVs.

PVs in this documentation can be any Kubernetes supported Persistent Volume types. This document uses NFS as an example PV type.

Note:

The backup method described in this document is supported starting from TiDB Operator v1.1.8.

6.6.5.1.1 Ad-hoc backup

Ad-hoc backup supports both full backup and incremental backup. It describes the backup by creating a `Backup` Custom Resource (CR) object. TiDB Operator performs the specific backup operation based on this `Backup` object. If an error occurs during the backup process, TiDB Operator does not retry, and you need to handle this error manually.

This document provides examples in which the data of the `demo1` TiDB cluster in the `test1` Kubernetes namespace is backed up to NFS.

Prerequisites for ad-hoc backup

Note:

If TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR will automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.tikvGCLifeTime` and `spec.from` fields in the `Backup` CR. In addition, you can skip the steps of creating the `backup-demo1-tidb-secret` secret and configuring database account privileges.

1. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `test1` namespace:

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

2. Create the `backup-demo1-tidb-secret` secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-literal=
    ↪ password=<password> --namespace=test1
```

3. Ensure that the NFS server is accessible from your Kubernetes cluster, and TiKV is configured to mount the same NFS server directory to the same local path as in backup jobs. To mount NFS for TiKV, refer to the configuration below:

```
spec:
  tikv:
```

```
  additionalVolumes:
  # specify volume types that are supported by Kubernetes, Ref: https
    ↪ ://kubernetes.io/docs/concepts/storage/persistent-volumes/#
    ↪ types-of-persistent-volumes
  - name: nfs
    nfs:
      server: 192.168.0.2
      path: /nfs
  additionalVolumeMounts:
  # this must match `name` in `additionalVolumes`
  - name: nfs
    mountPath: /nfs
```

Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the backup, the `Backup` CR needs a database account with these privileges to adjust the GC time.

Process of ad-hoc backup

1. Create the `Backup` CR, and back up cluster data to NFS as described below:

```
kubectl apply -f backup-nfs.yaml
```

The content of `backup-nfs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  # from:
  #   host: ${tidb-host}
  #   port: ${tidb-port}
  #   user: ${tidb-user}
  #   secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
```

```
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
    # options:
    # - --lastbackupts=420134118382108673
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

In the example above, `spec.local` refers to the configuration related to PVs. For more information about PV configuration, refer to Local storage fields.

In the example above, some parameters in `spec.br` can be ignored, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, and `timeAgo`. For more information about BR configuration, refer to BR fields.

Since TiDB Operator v1.1.6, if you want to back up data incrementally, you only need to specify the last backup timestamp `--lastbackupts` in `spec.br.options`. For the limitations of incremental backup, refer to Use BR to Back up and Restore Data.

For more information about the `Backup` CR fields, refer to Backup CR fields.

This example backs up all data in the TiDB cluster to NFS.

2. After creating the `Backup` CR, use the following command to check the backup status:

```
kubectl get bk -n test1 -owide
```

Backup CR examples

Back up data of all clusters

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
```

```
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  br:
    cluster: demo1
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

Back up data of a single database

The following example backs up data of the `db1` database.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
```

```
    name: nfs
    nfs:
      server: ${nfs_server_ip}
      path: /nfs
  volumeMount:
    name: nfs
    mountPath: /nfs
```

Back up data of a single table

The following example backs up data of the `db1.table1` table.

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

Back up data of multiple tables using the table filter

The following example backs up data of the `db1.table1` table and `db1.table2` table.

```
---
```

```
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-nfs
  namespace: test1
spec:
  # # backupType: full
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  from:
    host: ${tidb-host}
    port: ${tidb-port}
    user: ${tidb-user}
    secretName: backup-demo1-tidb-secret
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  br:
    cluster: demo1
    clusterNamespace: test1
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_ip}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

6.6.5.1.2 Scheduled full backup

You can set a backup policy to perform scheduled backups of the TiDB cluster, and set a backup retention policy to avoid excessive backup items. A scheduled full backup is described by a custom `BackupSchedule` CR object. A full backup is triggered at each backup time point. Its underlying implementation is the ad-hoc full backup.

Prerequisites for scheduled full backup

The prerequisites for the scheduled full backup is the same with the prerequisites for ad-hoc backup.

Process of scheduled full backup

1. Create the `BackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f backup-schedule-nfs.yaml
```

The content of `backup-schedule-nfs.yaml` is as follows:

```
  yaml  ---  apiVersion: pingcap.com/v1alpha1 kind: BackupSchedule
↪ metadata:  name: demo1-backup-schedule-nfs  namespace: test1  spec:
↪  #maxBackups: 5  #pause: true  maxReservedTime: "3h"  schedule
↪ : "*/2 * * * *"  backupTemplate:  # Only needed for TiDB Operator
↪  < v1.1.10 or TiDB < v4.0.8 # from:  #  host: ${tidb_host}  #
↪  port: ${tidb_port}  #  user: ${tidb_user}  #  secretName:
↪ backup-demo1-tidb-secret  br:  cluster: demo1  clusterNamespace
↪ : test1  # logLevel: info  # statusAddr: ${status-addr}  #
↪  concurrency: 4  # rateLimit: 0  # checksum: true  local:
↪  prefix: backup-nfs  volume:  name: nfs  nfs
↪ :  server: ${nfs_server_ip}  path: /nfs  volumeMount
↪ :  name: nfs  mountPath: /nfs
```

2. After creating the scheduled full backup, use the following command to check the backup status:

```
kubectl get bks -n test1 -owide
```

Use the following command to check all the backup items:

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

From the example above, you can see that the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`.

`backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR. For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.

6.6.5.1.3 Delete the backup CR

Refer to Delete the Backup CR.

6.6.5.1.4 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

6.6.5.2 Restore Data from PV Using BR

This document describes how to restore the TiDB cluster data backed up using TiDB Operator in Kubernetes. BR is used to perform the restore.

PVs in this documentation can be any Kubernetes supported Persistent Volume types. This document uses NFS as an example PV type, and shows an example in which the backup data stored in the specified path on NFS is restored to the TiDB cluster.

The restore method described in this document is implemented based on Custom Resource Definition (CRD) in TiDB Operator v1.1 or later versions.

6.6.5.2.1 Prerequisites

> **Note:**
>
> If TiDB Operator >= v1.1.10 && TiDB >= v4.0.8, BR will automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.to` fields in the `Restore` CR. In addition, you can skip the steps of creating the `restore -demo2-tidb-secret` secret and configuring database account privileges.

1. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `test2` namespace:

```
kubectl apply -f backup-rbac.yaml -n test2
```

2. Create the `restore-demo2-tidb-secret` secret which stores the root account and password needed to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-literal=
    ↪ user=root --from-literal=password=<password> --namespace=test2
```

3. Ensure that the NFS server is accessible from your Kubernetes cluster.

6.6.5.2.2 Required database account privileges

- The `SELECT` and `UPDATE` privileges of the `mysql.tidb` table: Before and after the restore, the `Restore` CR needs a database account with these privileges to adjust the GC time.

6.6.5.2.3 Restore process

1. Create the `Restore` custom resource (CR), and restore the specified data to your cluster:

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

The content of `restore.yaml` file is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-nfs
  namespace: test2
spec:
  # backupType: full
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
  # # Only needed for TiDB Operator < v1.1.10 or TiDB < v4.0.8
  # to:
  #   host: ${tidb_host}
  #   port: ${tidb_port}
  #   user: ${tidb_user}
  #   secretName: restore-demo2-tidb-secret
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_if}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

2. After creating the `Restore` CR, execute the following command to check the restore status:

```
kubectl get rt -n test2 -owide
```

The example above restores data from the `local://${.spec.local.volumeMount.mountPath}/${.spec.local.prefix}/` directory on NFS to the `demo2` TiDB cluster in the `test2` namespace. For more information about local storage configuration, refer to Local storage fields.

In the example above, some parameters in `spec.br` can be ignored, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.

For more information about the `Restore` CR fields, refer to Restore CR fields.

6.6.5.2.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

6.7 Restart a TiDB Cluster in Kubernetes

If you find that the memory leak occurs in a Pod during use, you need to restart the cluster. This document describes how to perform a graceful rolling restart to all the Pods in a component of the TiDB cluster, or gracefully log off a Pod in the TiDB cluster and then restart the Pod using the graceful restart command.

> **Warning:**
>
> It is not recommended to manually remove a Pod in the TiDB cluster without graceful restart in a production environment, because this might lead to some request failures of accessing the TiDB cluster though the `StatefulSet` controller pulls the Pod up again.

6.7.1 Performing a graceful rolling restart to all Pods in a component

After Deploying TiDB on general Kubernetes, modify the cluster configuration by running the following command:

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

Add `tidb.pingcap.com/restartedAt` in the annotation of the `spec` of the TiDB component you want to gracefully rolling restart, and set its value to be the current time.

In the following example, annotations of the `pd`, `tikv`, and `tidb` components are set, which means that all the Pods in these three components will be gracefully rolling restarted. You can set the annotation for a specific component according to your needs.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: basic
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
  tikv:
    baseImage: pingcap/tikv
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
    config: {}
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
```

6.8 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 PD, TiKV, and TiDB Pods because of their different characteristics.

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

6.8.1 Prerequisites

- kubectl
- tkctl
- jq

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

6.8.2 Maintain a node that can be recovered shortly

1. Mark the node to be maintained as non-schedulable to ensure that no new Pod is scheduled to it:

```
kubectl cordon ${node_name}
```

2. Check whether there is any TiKV Pod on the node to be maintained:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

   If any TiKV Pod is found, for each TiKV Pod, perform the following operations:

   1. Evict the TiKV Region Leader to another Pod.
   2. Increase the maximum offline duration for TiKV Pods by configuring `max-store-down-time` of PD. After you maintain and recover the Kubernetes node within that duration, all TiKV Pods on that node will be automatically recovered.

      The following example shows how to set `max-store-down-time` to `60m`. You can set it to any reasonable value.

      ```
      pd-ctl config set max-store-down-time 60m
      ```

3. Check whether there is any PD Pod on the node to be maintained:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

   If any PD Pod is found, for each PD Pod, transfer the PD leader to other Pods.

4. Confirm that the node to be maintained no longer has any TiKV Pod or PD Pod:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

5. Migrate all Pods on the node to be maintained to other nodes:

```
kubectl drain ${node_name} --ignore-daemonsets
```

After running this command, all Pods on this node are automatically migrated to another available node.

6. Confirm that the node to be maintained no longer has any TiKV, TiDB, or PD Pod:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

7. When the maintenance is completed, after you recover the node, make sure that the node is in a healthy state:

```
watch kubectl get node ${node_name}
```

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

8. Lift the scheduling restriction on the node:

```
kubectl uncordon ${node_name}
```

9. Confirm that all Pods are running normally:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

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

6.8.3 Maintain a node that cannot be recovered shortly

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

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

If any TiKV Pod is found, for each TiKV Pod, reschedule the TiKV Pod to another node.

2. Check whether there is any PD Pod on the node to be maintained:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

If any PD Pod is found, for each PD Pod, reschedule the PD Pod to another node.

3. Confirm that the node to be maintained no longer has any TiKV Pod or PD Pod:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

4. Migrate all Pods on the node to be maintained to other nodes:

```
kubectl drain ${node_name} --ignore-daemonsets
```

   After running this command, all Pods on this node are automatically migrated to another available node.

5. Confirm that the node to be maintained no longer has any TiKV, TiDB, or PD Pod:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name}
```

6. (Optional) If the node will be offline for a long time, it is recommended to delete the node from your Kubernetes cluster:

```
kubectl delete node ${node_name}
```

6.8.4 Reschedule PD Pods

If a node will be offline for a long time, to minimize the impact on your application, you can reschedule the PD Pods on this node to other nodes in advance.

6.8.4.1 If the node storage can be automatically migrated

If the node storage can be automatically migrated (such as EBS), to reschedule a PD Pod, you do not need to delete the PD member. You only need to transfer the PD Leader to another Pod and delete the old Pod.

1. Mark the node to be maintained as non-schedulable to ensure that no new Pod is scheduled to it:

```
kubectl cordon ${node_name}
```

2. Check the PD Pod on the node to be maintained:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

3. Transfer the PD Leader to another Pod.

4. Delete the old PD Pod:

```
kubectl delete -n ${namespace} pod ${pod_name}
```

5. Confirm that the PD Pod is successfully scheduled to another node:

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

6.8.4.2 If the node storage cannot be automatically migrated

If the node storage cannot be automatically migrated (such as local storage), to reschedule a PD Pod, you need to delete the PD member.

1. Mark the node to be maintained as non-schedulable to ensure that no new Pod is scheduled to it:

```
kubectl cordon ${node_name}
```

2. Check the PD Pod on the node to be maintained:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep pd
```

3. Transfer the PD Leader to another Pod.

4. Take the PD Pod offline:

```
pd-ctl member delete name ${pod_name}
```

5. Confirm that the PD member is deleted:

```
pd-ctl member
```

6. Unbind the PD Pod with the local disk on the node.

    1. Check the `PersistentVolumeClaim` used by the Pod:

```
kubectl -n ${namespace} get pvc -l tidb.pingcap.com/pod-name=${
    ↪ pod_name}
```

    2. Delete the `PersistentVolumeClaim`:

```
kubectl delete -n ${namespace} pvc ${pvc_name} --wait=false
```

7. Delete the old PD Pod:

```
kubectl delete -n ${namespace} pod ${pod_name}
```

8. Confirm that the PD Pod is successfully scheduled to another node:

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

6.8.5 Reschedule TiKV Pods

If a node will be offline for a long time, to minimize the impact on your application, you can reschedule the TiKV Pods on this node to other nodes in advance.

6.8.5.1 If the node storage can be automatically migrated

If the node storage can be automatically migrated (such as EBS), to reschedule a TiKV Pod, you do not need to delete the whole TiKV store. You only need to evict the TiKV Region Leader to another Pod and delete the old Pod.

1. Mark the node to be maintained as non-schedulable to ensure that no new Pod is scheduled to it:

```
kubectl cordon ${node_name}
```

2. Check the TiKV Pod on the node to be maintained:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

3. Evict the TiKV Region Leader to another Pod.

4. Delete the old TiKV Pod:

```
kubectl delete -n ${namespace} pod ${pod_name}
```

5. Confirm that the TiKV Pod is successfully scheduled to another node:

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

6. Remove evict-leader-scheduler, and wait for the Region Leader to automatically schedule back:

```
pd-ctl scheduler remove evict-leader-scheduler-${ID}
```

6.8.5.2 If the node storage cannot be automatically migrated

If the node storage cannot be automatically migrated (such as local storage), to reschedule a TiKV Pod, you need to delete the whole TiKV store.

1. Mark the node to be maintained as non-schedulable to ensure that no new Pod is scheduled to it:

```
kubectl cordon ${node_name}
```

2. Check the TiKV Pod on the node to be maintained:

```
kubectl get pod --all-namespaces -o wide | grep ${node_name} | grep
    ↪ tikv
```

3. Evict the TiKV Region Leader to another Pod.

4. Take the TiKV Pod offline.

> **Note:**
> Before you take the TiKV Pod offline, make sure that the remaining TiKV Pods are not fewer than the TiKV replica number set in PD configuration (`max-replicas`, 3 by default). If the remaining TiKV Pods are not enough, scale out TiKV Pods before you take the TiKV Pod offline.

   1. Check `store-id` of the TiKV Pod:

```
kubectl get tc ${cluster_name} -ojson | jq ".status.tikv.stores |
    ↪ .[] | select ( .podName == \"${pod_name}\" ) | .id"
```

   2. Take the Pod offline:

```
pd-ctl store delete ${ID}
```

5. Wait for the store status (`state_name`) to become `Tombstone`:

```
watch pd-ctl store ${ID}
```

6. Unbind the TiKV Pod with the local disk on the node.

   1. Check the `PersistentVolumeClaim` used by the Pod:

```
kubectl -n ${namespace} get pvc -l tidb.pingcap.com/pod-name=${
    ↪ pod_name}
```

   2. Delete the `PersistentVolumeClaim`:

```
kubectl delete -n ${namespace} pvc ${pvc_name} --wait=false
```

7. Delete the old TiKV Pod:

```
kubectl delete -n ${namespace} pod ${pod_name}
```

8. Confirm that the TiKV Pod is successfully scheduled to another node:

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

9. Remove evict-leader-scheduler, and wait for the Region Leader to automatically schedule back:

```
pd-ctl scheduler remove evict-leader-scheduler-${ID}
```

6.8.6 Transfer PD Leader

1. Check the PD Leader:

```
pd-ctl member leader show
```

2. If the Leader Pod is on the node to be maintained, you need to transfer the PD Leader to a Pod on another node:

```
pd-ctl member leader transfer ${pod_name}
```

`${pod_name}` is the name of the PD Pod on another node.

6.8.7 Evict TiKV Region Leader

1. Check `store-id` of the TiKV Pod:

```
kubectl get tc ${cluster_name} -ojson | jq ".status.tikv.stores | .[] |
    ↪ select ( .podName == \"${pod_name}\" ) | .id"
```

2. Evict the Region Leader:

```
pd-ctl scheduler add evict-leader-scheduler ${ID}
```

3. Confirm that all Region Leaders are migrated:

```
kubectl get tc ${cluster_name} -ojson | jq ".status.tikv.stores | .[] |
    ↪ select ( .podName == \"${pod_name}\" ) | .leaderCount"
```

6.9 View TiDB Logs in Kubernetes

This document introduces the methods to view logs of TiDB components and TiDB slow log.

6.9.1 View logs of TiDB components

The TiDB components deployed by TiDB Operator output the logs in the `stdout` and `stderr` of the container by default. You can view the log of a single Pod by running the following command:

```
kubectl logs -n ${namespace} ${pod_name}
```

If the Pod has multiple containers, you can also view the logs of a container in this Pod:

```
kubectl logs -n ${namespace} ${pod_name} -c ${container_name}
```

For more methods to view Pod logs, run `kubectl logs --help`.

6.9.2 View slow query logs of TiDB components

For TiDB 3.0 or later versions, TiDB separates slow query logs from application logs. You can view slow query logs from the sidecar container named `slowlog`:

```
kubectl logs -n ${namespace} ${pod_name} -c slowlog
```

> **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 might be different. For this reason, the tool for parsing MySQL slow query logs may not be fully compatible with TiDB slow query logs.

6.10 Automatic failover

TiDB Operator manages the deployment and scaling of Pods based on `StatefulSet`. When some Pods or nodes fail, `StatefulSet` does not support automatically creating new Pods to replace the failed ones. To solve this issue, TiDB Operator supports the automatic failover feature by scaling Pods automatically.

6.10.1 Configure automatic failover

The automatic failover feature is enabled by default in TiDB Operator.

When deploying TiDB Operator, you can configure the waiting timeout for failover of the PD, TiKV, TiDB, and TiFlash components in a TiDB cluster in the `charts/tidb-operator/values.yaml` file. An example is as follows:

```
controllerManager:
 ...
 # autoFailover is whether tidb-operator should auto failover when failure occurs
 autoFailover: true
 # pd failover period default(5m)
 pdFailoverPeriod: 5m
 # tikv failover period default(5m)
 tikvFailoverPeriod: 5m
 # tidb failover period default(5m)
 tidbFailoverPeriod: 5m
 # tiflash failover period default(5m)
 tiflashFailoverPeriod: 5m
```

In the example, `pdFailoverPeriod`, `tikvFailoverPeriod`, `tiflashFailoverPeriod` ↪ and `tidbFailoverPeriod` indicate the waiting timeout (5 minutes by default) after an instance failure is identified. After the timeout, TiDB Operator starts the automatic failover process.

In addition, when configuring a TiDB cluster, you can specify `spec.${component}.` ↪ `maxFailoverCount` for each component, which is the threshold of the maximum number of Pods that the TiDB Operator can create during automatic failover. For more information, see the TiDB component configuration documentation.

> **Note:**
>
> If there are not enough resources in the cluster for TiDB Operator to create new Pods, the newly scaled Pods will be in the pending status.

6.10.2 Automatic failover policies

There are six components in a TiDB cluster: PD, TiKV, TiDB, TiFlash, TiCDC, and Pump. Currently, TiCDC and Pump do not support the automatic failover feature. PD, TiKV, TiDB, and TiFlash have different failover policies. This section gives a detailed introduction to these policies.

6.10.2.1 Failover with PD

TiDB Operator collects the health status of PD members via the `pd/health` PD API and records the status in the `.status.pd.members` field of the TidbCluster CR.

Take a PD cluster with 3 Pods as an example. If a Pod fails for more than 5 minutes (`pdFailoverPeriod` is configurable), TiDB Operator automatically does the following operations:

1. TiDB Operator records the Pod information in the `.status.pd.failureMembers` field of TidbCluster CR.
2. TiDB Operator takes the Pod offline: TiDB Operator calls PD API to remove the Pod from the member list, and then deletes the Pod and its PVC.
3. The StatefulSet controller recreates the Pod, and the recreated Pod joins the cluster as a new member.
4. When calculating the replicas of PD StatefulSet, TiDB Operator takes the deleted `.status.pd.failureMembers` into account, so it will create a new Pod. Then, 4 Pods will exist at the same time.

When all the failed Pods in the cluster recover, TiDB Operator will automatically remove the newly created Pods, and the number of Pods gets back to the original.

Note:

- For each PD cluster, the maximum number of Pods that TiDB Operator can create is `spec.pd.maxFailoverCount` (the default value is `3`). After the threshold is reached, TiDB Operator will not perform failover.
- If most members in a PD cluster fail, which makes the PD cluster unavailable, TiDB Operator will not perform failover for the PD cluster.

6.10.2.2 Failover with TiDB

TiDB Operator collects the Pod health status by accessing the `/status` interface of each TiDB Pod and records the status in the `.status.tidb.members` field of the TidbCluster CR.

Take a TiDB cluster with 3 Pods as an example. If a Pod fails for more than 5 minutes (`tidbFailoverPeriod` is configurable), TiDB Operator automatically does the following operations:

1. TiDB Operator records the Pod information in the `.status.tidb.failureMembers` field of TidbCluster CR.
2. When calculating the replicas of TiDB StatefulSet, TiDB Operator takes the `.status` ↪ `.tidb.failureMembers` into account, so it will create a new Pod. Then, 4 Pods will exist at the same time.

When the failed Pod in the cluster recovers, TiDB Operator will automatically remove the newly created Pod, and the number of Pods gets back to 3.

Note:

For each TiDB cluster, the maximum number of Pods that TiDB Operator can create is `spec.tidb.maxFailoverCount` (the default value is `3`). After the threshold is reached, TiDB Operator will not perform failover.

6.10.2.3 Failover with TiKV

TiDB Operator collects the TiKV store health status by accessing the PD API and records the status in the `.status.tikv.stores` field in TidbCluster CR.

Take a TiKV cluster with 3 Pods as an example. When a TiKV Pod fails, the store status of the Pod changes to `Disconnected`. By default, after 30 minutes (configurable

by changing `max-store-down-time = "30m"` in the `[schedule]` section of `pd.config`), the status changes to `Down`. Then, TiDB Operator automatically does the following operations:

1. Wait for another 5 minutes (configurable by modifying `tikvFailoverPeriod`), if this TiKV Pod is still not recovered, TiDB Operator records the Pod information in the `.status.tikv.failureStores` field of TidbCluster CR.
2. When calculating the replicas of TiKV StatefulSet, TiDB Operator takes the `.status ↪ .tikv.failureStores` into account, so it will create a new Pod. Then, 4 Pods will exist at the same time.

When the failed Pod in the cluster recovers, TiDB Operator **DOES NOT** remove the newly created Pod, but continues to keep 4 Pods. This is because scaling in TiKV Pods will trigger data migration, which might affect the cluster performance.

> **Note:**
>
> For each TiKV cluster, the maximum number of Pods that TiDB Operator can create is `spec.tikv.maxFailoverCount` (the default value is `3`). After the threshold is reached, TiDB Operator will not perform failover.

If **all** failed Pods have recovered, and you want to remove the newly created Pods, you can follow the procedure below:

Configure `spec.tikv.recoverFailover: true` (Supported since TiDB Operator v1.1.5):

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

TiDB Operator will remove the newly created Pods automatically. When the removal is finished, configure `spec.tikv.recoverFailover: false` to avoid the auto-scaling operation when the next failover occurs and recovers.

6.10.2.4 Failover with TiFlash

TiDB Operator collects the TiFlash store health status by accessing the PD API and records the status in the `.status.tiflash.stores` field in TidbCluster CR.

Take a TiFlash cluster with 3 Pods as an example. When a TiFlash Pod fails, the store status of the Pod changes to `Disconnected`. By default, after 30 minutes (configurable by changing `max-store-down-time = "30m"` in the `[schedule]` section of `pd.config`), the status changes to `Down`. Then, TiDB Operator automatically does the following operations:

1. Wait for another 5 minutes (configurable by modifying `tiflashFailoverPeriod`), if the TiFlash Pod is still not recovered, TiDB Operator records the Pod information in the `.status.tiflash.failureStores` field of TidbCluster CR.

2. When calculating the replicas of TiFlash StatefulSet, TiDB Operator takes the `.status ↪ .tiflash.failureStores` into account, so it will create a new Pod. Then, 4 Pods will exist at the same time.

When the failed Pod in the cluster recovers, TiDB Operator **DOES NOT** remove the newly created Pod, but continues to keep 4 Pods. This is because scaling in TiFlash Pods will trigger data migration, which might affect the cluster performance.

> **Note:**
>
> For each TiFlash cluster, the maximum number of Pods that TiDB Operator can create is `spec.tiflash.maxFailoverCount` (the default value is `3`). After the threshold is reached, TiDB Operator will not perform failover.

If **all** of the failed Pods have recovered, and you want to remove the newly created Pods, you can follow the procedure below:

Configure `spec.tiflash.recoverFailover: true` (Supported since TiDB Operator v1.1.5):

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

TiDB Operator will remove the newly created Pods automatically. When the removal is finished, configure `spec.tiflash.recoverFailover: false` to avoid the auto-scaling operation when the next failover occurs and recovers.

6.10.2.5 Disable automatic failover

You can disable the automatic failover feature at the cluster or component level:

- To disable the automatic failover feature at the cluster level, set `controllerManager ↪ .autoFailover` to `false` in the `charts/tidb-operator/values.yaml` file when deploying TiDB Operator. An example is as follows:

  ```
  controllerManager:
  ...
  # autoFailover is whether tidb-operator should auto failover when
      ↪ failure occurs
  autoFailover: false
  ```

- To disable the automatic failover feature at the component level, set `spec.${ ↪ component}.maxFailoverCount` of the target component to `0` in the TidbCluster CR when creating the TiDB cluster.

6.11 Destroy TiDB Clusters in Kubernetes

This document describes how to destroy TiDB clusters in Kubernetes.

6.11.1 Destroy a TiDB cluster managed by `TidbCluster`

To destroy a TiDB cluster managed by `TidbCluster`, run the following command:

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

If you deploy the monitor in the cluster using `TidbMonitor`, run the following command to delete the monitor component:

```
kubectl delete tidbmonitor ${tidb_monitor_name} -n ${namespace}
```

6.11.2 Destroy a TiDB cluster managed by Helm

To destroy a TiDB cluster managed by Helm, run the following command:

```
helm uninstall ${cluster_name} -n ${namespace}
```

6.11.3 Delete data

The above commands that destroy the cluster only remove the running Pod, but the data is still retained. If you want to delete the data as well, use the following commands:

> **Warning:**
>
> The following commands delete your data completely. Please be cautious.
>
> To ensure data safety, do not delete PVs on any circumstances, unless you are familiar with the working principles of PVs.

```
kubectl delete pvc -n ${namespace} -l app.kubernetes.io/instance=${
    ↪ cluster_name},app.kubernetes.io/managed-by=tidb-operator
```

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

6.12 Migrate from Helm 2 to Helm 3

This document describes how to migrate component management from Helm 2 to Helm 3. This document takes TiDB Operator as an example. For other components, you can take similar steps to perform the migration.

For more information about migrating releases managed by Helm 2 to Helm 3, refer to Helm documentation.

6.12.1 Migration procedure

In this example, TiDB Operator (`tidb-operator`) managed by Helm 2 is installed in the `tidb-admin` namespace. A `basic` TidbCluster and `basic` TidbMonitor are deployed in the `tidb-cluster` namespace.

```
helm list
```

```
NAME          REVISION      UPDATED                       STATUS        CHART
   ↪                  APP VERSION   NAMESPACE
tidb-operator 1             Tue Jan 5 15:28:00 2021     DEPLOYED      tidb-
   ↪ operator-v1.1.8 v1.1.8        tidb-admin
```

1. Install Helm 3.

   Helm 3 takes a different approach from Helm 2 in configuration and data storage. Therefore, when you install Helm 3, there is no risk of overwriting the original configuration and data.

   > **Note:**
   > During the installation, do not let the CLI binary of Helm 3 overwrite that of Helm 2. For example, you can name Helm 3 CLI binary as `helm3`. (All following examples in this document uses `helm3`.)

2. Install helm-2to3 plugin for Helm 3.

   ```
   helm3 plugin install https://github.com/helm/helm-2to3
   ```

   You can verify whether the plugin is successfully installed by running the following command:

   ```
   helm3 plugin list
   ```

   ```
   NAME   VERSION DESCRIPTION
   2to3   0.8.0  migrate and cleanup Helm v2 configuration and releases in
      ↪ -place to Helm v3
   ```

3. Migrate the configuration such as repo and plugins from Helm 2 to Helm 3:

```
helm3 2to3 move config
```

Before you migrate, you can learn about the possible operations and their consequences by executing `helm3 2to3 move config --dry-run`.

After the migration, the PingCAP repo is already in Helm 3:

```
helm3 repo list
```

```
NAME   URL
pingcap https://charts.pingcap.org/
```

4. Migrate the releases from Helm 2 to Helm 3:

```
helm3 2to3 convert tidb-operator
```

Before you migrate, you can learn about the possible operations and their consequences by executing `helm3 2to3 convert tidb-operator --dry-run`.

After the migration, you can view the release corresponding to TiDB Operator in Helm 3:

```
helm3 list --namespace=tidb-admin
```

```
NAME          NAMESPACE     REVISION      UPDATED
   ↪                                STATUS        CHART                 APP
   ↪  VERSION
tidb-operator tidb-admin    1             2021-01-05 07:28:00.3545941
   ↪ +0000 UTC deployed   tidb-operator-v1.1.8 v1.1.8
```

> **Note:**
> If the original Helm 2 is Tillerless (Tiller is installed locally via plugins like helm-tiller rather than in the Kubernetes cluster), you can migrate the release by adding the `--tiller-out-cluster` flag in the command: `helm3 2to3 convert tidb-operator --tiller-out-cluster`.

5. Confirm that TiDB Operator, TidbCluster, and TidbMonitor run normally.

To view the running status of TiDB Operator:

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

If all Pods are in the Running state, TiDB Operator runs normally:

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

To view the running status of TidbCluster and TidbMonitor:

```
watch kubectl get pods --namespace=tidb-cluster
```

If all Pods are in the Running state, TidbCluster and TidbMonitor runs normally:

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

6. Clean up Helm 2 data, such as configuration and releases:

```
helm3 2to3 cleanup --name=tidb-operator
```

Before you clean up the data, you can learn about the possible operations and their consequences by executing `helm3 2to3 cleanup --name=tidb-operator --dry-run` ↪ .

> **Note:**
> After the cleanup, you can no longer manage the releases using Helm 2.

7 Disaster Recovery

7.1 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.

7.1.1 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 `v5.0.6`.

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.

7.1.2 Recover the PD cluster

This section introduces how to recover the PD cluster using PD Recover.

7.1.2.1 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
```

7.1.2.2 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 Prometheus 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`.

7.1.2.3 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 cluster is started:

```
kubectl logs -f ${cluster_name}-pd-0 -n ${namespace} | grep "Welcome to
    ↪  Placement Driver (PD)"
```

7.1.2.4 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.

7.1.2.5 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.

7.1.2.6 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}
```

7.1.2.7 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}
```

7.2 Recover the Deleted Cluster

This document describes how to recover a TiDB cluster that has been deleted mistakenly.

7.2.1 Recover the cluster managed by TidbCluster

TiDB Operator uses PV (Persistent Volume) and PVC (Persistent Volume Claim) to store persistent data. If you accidentally delete a cluster using `kubectl delete tc`, the PV/PVC objects and data are still retained to ensure data safety.

To recover the deleted cluster, use the `kubectl create` command to create a cluster that has the same name and configuration as the deleted one. In the new cluster, the retained PV/PVC and data are reused.

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

7.2.2 Recover the cluster managed by Helm

TiDB Operator uses PV (Persistent Volume) and PVC (Persistent Volume Claim) to store persistent data. If you accidentally delete a cluster using `helm uninstall`, the PV/PVC objects and data are still retained to ensure data safety.

To recover the cluster at this time, use the `helm install` command to create a cluster that has the same name and configuration as the deleted one. In the new cluster, the retained PV/PVC and data are reused.

```
helm install ${release_name} pingcap/tidb-cluster --namespace=${namespace}
    ↪ --version=${chart_version} -f values.yaml
```

8 Import Data

This document describes how to import 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 separate 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`.

TiDB Lightning supports three backends: `Importer-backend`, `Local-backend`, and `TiDB-backend`. For the differences of these backends and how to choose backends, see TiDB Lightning Backends.

- For `Importer-backend`, both tikv-importer and tidb-lightning need to be deployed.
- For `Local-backend`, only tidb-lightning needs to be deployed.
- For `TiDB-backend`, only tidb-lightning needs to be deployed, and it is recommended to import data using CustomResourceDefinition (CRD) in TiDB Operator v1.1 and later versions. For details, refer to Restore Data from GCS Using TiDB Lightning or Restore Data from S3-Compatible Storage Using TiDB Lightning

8.1 Deploy TiKV Importer

> **Note:**
>
> If you use the `local` or `tidb` backend for data restore, you can skip deploying tikv-importer and deploy tidb-lightning directly.

You can deploy tikv-importer using the Helm chart. See the following example:

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

```
helm repo update
```

```
helm search repo tikv-importer -l
```

2. Get the default `values.yaml` file for easier customization:

```
helm inspect values pingcap/tikv-importer --version=${chart_version} >
    ↪ values.yaml
```

3. Modify the `values.yaml` file to specify the target TiDB cluster. See the following example:

```
clusterName: demo
image: pingcap/tidb-lightning:v5.0.6
imagePullPolicy: IfNotPresent
storageClassName: local-storage
storage: 20Gi
pushgatewayImage: prom/pushgateway:v0.3.1
pushgatewayImagePullPolicy: IfNotPresent
config: |
  log-level = "info"
  [metric]
  job = "tikv-importer"
  interval = "15s"
  address = "localhost:9091"
```

`clusterName` must match the target TiDB cluster.

If the target TiDB cluster has enabled TLS between components (`spec.tlsCluster.enabled: true`), refer to Generate certificates for components of the TiDB cluster to genereate a server-side certificate for TiKV Importer, and configure `tlsCluster.enabled: true` in `values.yaml` to enable TLS.

4. Deploy tikv-importer:

```
helm install ${cluster_name} pingcap/tikv-importer --namespace=${
    ↪ namespace} --version=${chart_version} -f values.yaml
```

Note:

You must deploy tikv-importer in the same namespace where the target TiDB cluster is deployed.

8.2 Deploy TiDB Lightning

8.2.1 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
```

Configure a `backend` used by TiDB Lightning depending on your needs. To do that, you can set the `backend` value in `values.yaml` to an option in `importer`, `local`, or `tidb`.

> **Note:**
>
> If you use the `local` backend, you must set `sortedKV` in `values.yaml` to create the corresponding PVC. The PVC is used for local KV sorting.

Starting from v1.1.10, the tidb-lightning Helm chart saves the TiDB Lightning checkpoint information in the directory of the source data. When the a new tidb-lightning job is running, it can resume the data import according to the checkpoint information.

For versions earlier than v1.1.10, you can modify `config` in `values.yaml` to save the checkpoint information in the target TiDB cluster, other MySQL-compatible databases or a shared storage directory. For more information, refer to TiDB Lightning checkpoint.

If TLS between components has been enabled on the target TiDB cluster (`spec.tlsCluster.enabled: true`), refer to Generate certificates for components of the TiDB cluster to genereate a server-side certificate for TiDB Lightning, and configure `tlsCluster.enabled: true` in `values.yaml` to enable TLS between components.

If the target TiDB cluster has enabled TLS for the MySQL client (`spec.tidb.tlsClient.enabled: true`), and the corresponding client-side certificate is configured (the Kubernetes Secret object is `${cluster_name}-tidb-client-secret`), you can configure `tlsClient.enabled: true` in `values.yaml` to enable TiDB Lightning to connect to the TiDB server using TLS.

To use different client certificates to connect to the TiDB server, refer to Issue two sets of certificates for the TiDB cluster to generate the client-side certificate for TiDB Lightning, and configure the corresponding Kubernetes secret object in `tlsCluster.tlsClientSecretName` in `values.yaml`.

> **Note:**

If TLS is enabled between components via `tlsCluster.enabled: true` but not enabled between TiDB Lightning and the TiDB server via `tlsClient.enabled: true`, you need to explicitly disable TLS between TiDB Lightning and the TiDB server in `config` in `values.yaml`:

```
[tidb]
tls="false"
```

TiDB Lightning Helm chart supports both local and remote data sources.

8.2.1.1 Local

In the local mode, the backup data must be on one of the Kubernetes node. To enable this mode, set `dataSource.local.nodeName` to the node name and `dataSource.local.hostPath` to the path of the backup data. The path should contain a file named `metadata`.

8.2.1.2 Remote

Unlike the local mode, the remote mode needs to use rclone to download the 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), Amazon S3, Ceph Object Storage.

To restore backup data from the remote source, take the following steps:

1. Make sure that `dataSource.local.nodeName` and `dataSource.local.hostPath` in `values.yaml` are commented out.

2. Grant permissions to the remote storage

   If you use Amazon S3 as the storage, refer to AWS account Permissions. The configuration varies with different methods.

   If you use Ceph as the storage, you can only grant permissions by importing AccessKey and SecretKey. See Grant permissions by AccessKey and SecretKey.

   If you use GCS as the storage, refer to GCS account permissions.

   - Grant permissions by importing AccessKey and SecretKey

     1. Create a `Secret` configuration file `secret.yaml` containing the rclone configuration. A sample configuration is listed below. Only one cloud storage configuration is required.

```
apiVersion: v1
kind: Secret
metadata:
  name: cloud-storage-secret
type: Opaque
stringData:
  rclone.conf: |
    [s3]
    type = s3
    provider = AWS
    env_auth = false
    access_key_id = ${access_key}
    secret_access_key = ${secret_key}
    region = us-east-1

    [ceph]
    type = s3
    provider = Ceph
    env_auth = false
    access_key_id = ${access_key}
    secret_access_key = ${secret_key}
    endpoint = ${endpoint}
    region = :default-placement

    [gcs]
    type = google cloud storage
    # The service account must include Storage Object Viewer
       ↪ role
    # The content can be retrieved by `cat ${service-account-
       ↪ file} | jq -c .`
    service_account_credentials = ${
       ↪ service_account_json_file_content}
```

2. Execute the following command to create `Secret`:

```
kubectl apply -f secret.yaml -n ${namespace}
```

- Grant permissions by associating IAM with Pod or with ServiceAccount

If you use Amazon S3 as the storage, you can grant permissions by associating IAM with Pod or with ServiceAccount, in which `s3.access_key_id` and `s3.secret_access_key` can be ignored.

```
1. Save the following configurations as `secret.yaml`.
```

```
    ```yaml
    apiVersion: v1
    kind: Secret
    metadata:
      name: cloud-storage-secret
    type: Opaque
    stringData:
      rclone.conf: |
        [s3]
        type = s3
        provider = AWS
        env_auth = true
        access_key_id =
        secret_access_key =
        region = us-east-1
    ```

2. Execute the following command to create `Secret`:


    ```shell
    kubectl apply -f secret.yaml -n ${namespace}
    ```
```

3. Configure the `dataSource.remote.storageClassName` to an existing storage class in the Kubernetes cluster.

8.2.1.3 Ad hoc

When restoring data from remote storage, sometimes the restore process is interrupted due to the exception. In such cases, if you do not want to download backup data from the network storage repeatedly, you can use the ad hoc mode to directly recover the data that has been downloaded and decompressed into PV in the remote mode. The steps are as follows:

1. Ensure `dataSource.local` and `dataSource.remote` in the config file `values.yaml` are empty。
2. Configure `dataSource.adhoc.pvcName` in `values.yaml` to the PVC name used in restoring data from remote storage.
3. Configure `dataSource.adhoc.backupName` in `values.yaml` to the name of the original backup data, such as: `backup-2020-12-17T10:12:51Z` (Do not contain the '. tgz' suffix of the compressed file name on network storage).

8.2.2 Deploy TiDB Lightning

The method of deploying TiDB Lightning varies with different methods of granting permissions and with different storages.

- For Local Mode, Ad hoc Mode, and Remote Mode (only for remote modes that meet one of the three requirements: using Amazon S3 AccessKey and SecretKey permission granting methods, using Ceph as the storage backend, or using GCS as the storage backend), run the following command to deploy TiDB Lightning.

```
helm install ${release_name} pingcap/tidb-lightning --namespace=${
    ↪ namespace} --set failFast=true -f tidb-lightning-values.yaml --
    ↪ version=${chart_version}
```

- For Remote Mode, if you grant permissions by associating Amazon S3 IAM with Pod, take the following steps:

  1. Create the IAM role:

     Create an IAM role for the account, and grant the required permission to the role. The IAM role requires the `AmazonS3FullAccess` permission because TiDB Lightning needs to access Amazon S3 storage.

  2. Modify `tidb-lightning-values.yaml`, and add the `iam.amazonaws.com/role ↪ : arn:aws:iam::123456789012:role/user` annotation in the `annotations` field.

  3. Deploy TiDB Lightning:

```
helm install ${release_name} pingcap/tidb-lightning --namespace=${
    ↪ namespace} --set failFast=true -f tidb-lightning-values.yaml
    ↪  --version=${chart_version}
```

> **Note:**
> `arn:aws:iam::123456789012:role/user` is the IAM role created in Step 1.

- For Remote Mode, if you grant permissions by associating Amazon S3 with ServiceAccount, take the following steps:

  1. Enable the IAM role for the service account on the cluster:

     To enable the IAM role permission on the EKS cluster, see AWS Documentation.

  2. Create the IAM role:

     Create an IAM role. Grant the `AmazonS3FullAccess` permission to the role, and edit `Trust relationships` of the role.

3. Associate IAM with the ServiceAccount resources:

```
kubectl annotate sa ${servieaccount} -n ${namespace} eks.amazonaws.
    ↪ com/role-arn=arn:aws:iam::123456789012:role/user
```

4. Deploy TiDB Lightning:

```
helm install ${release_name} pingcap/tidb-lightning --namespace=${
    ↪ namespace} --set-string failFast=true,serviceAccount=${
    ↪ servieaccount} -f tidb-lightning-values.yaml --version=${
    ↪ chart_version}
```

> **Note:**
> `arn:aws:iam::123456789012:role/user` is the IAM role created in Step 1. `${service-account}` is the ServiceAccount used by TiDB Lightning. The default value is `default`.

8.3 Destroy TiKV Importer and TiDB Lightning

Currently, TiDB Lightning only supports restoring data offline. After the restore, if the TiDB cluster needs to provide service for external applications, you can destroy TiDB Lightning to save cost.

To destroy tikv-importer, execute the following command:

```
helm uninstall ${release_name} -n ${namespace}
```

To destroy tidb-lightning, execute the following command:

```
helm uninstall ${release_name} -n ${namespace}
```

8.4 Troubleshoot TiDB Lightning

When TiDB Lightning fails to restore data, you cannot simply restart it. **Manual intervention** is required. Therefore, the TiDB Lightning's `Job` restart policy is set to `Never`.

> **Note:**
> If you have not configured to persist the checkpoint information in the target TiDB cluster, other MySQL-compatible databases or a shared storage directory, after the restore failure, you need to first delete the part of data already restored to the target cluster. After that, deploy tidb-lightning again and retry the data restore.

If TiDB Lightning fails to restore data, and if you have configured to persist the checkpoint information in the target TiDB cluster, other MySQL-compatible databases or a shared storage directory, follow the steps below to do manual intervention:

1. View the log by executing the following command:

```
kubectl logs -n ${namespace} ${pod_name}
```

   - If you restore data using the remote data source, and the error occurs when TiDB Lightning downloads data from remote storage:
     1. Address the problem according to the log.
     2. Deploy tidb-lightning again and retry the data restore.
   - For other cases, refer to the following steps.

2. Refer to TiDB Lightning Troubleshooting and learn the solutions to different issues.

3. Address the issues accordingly:

   - If `tidb-lightning-ctl` is required:
     1. Configure `dataSource` in `values.yaml`. Make sure the new `Job` uses the data source and checkpoint information of the failed `Job`:
        - In the local or ad hoc mode, you do not need to modify `dataSource`.
        - In the remote mode, modify `dataSource` to the ad hoc mode. `dataSource.adhoc.pvcName` is the PVC name created by the original Helm chart. `dataSource.adhoc.backupName` is the backup name of the data to be restored.
     2. Modify `failFast` in `values.yaml` to `false`, and create a `Job` used for `tidb-lightning-ctl`.
        - Based on the checkpoint information, TiDB Lightning checks whether the last data restore encountered an error. If yes, TiDB Lightning pauses the restore automatically.
        - TiDB Lightning uses the checkpoint information to avoid repeatedly restoring the same data. Therefore, creating the `Job` does not affect data correctness.
     3. After the Pod corresponding to the new `Job` is running, view the log by running `kubectl logs -n ${namespace} ${pod_name}` and confirm tidb-lightning in the new `Job` already stops data restore. If the log has the following message, the data restore is stopped:
        - `tidb lightning encountered error`
        - `tidb lightning exit`
     4. Enter the container by running `kubectl exec -it -n ${namespace} ${pod_name} -it -- sh`.

5. Obtain the starting script by running `cat /proc/1/cmdline`.
6. Get the command-line parameters from the starting script. Refer to TiDB Lightning Troubleshooting and troubleshoot using `tidb-lightning-ctl`.
7. After the troubleshooting, modify `failFast` in `values.yaml` to `true` and create a new `Job` to resume data restore.

- If `tidb-lightning-ctl` is not required:
  1. Troubleshoot TiDB Lightning.
  2. Configure `dataSource` in `values.yaml`. Make sure the new `Job` uses the data source and checkpoint information of the failed `Job`:
     - In the local or ad hoc mode, you do not need to modify `dataSource`.
     - In the remote mode, modify `dataSource` to the ad hoc mode. `dataSource.adhoc.pvcName` is the PVC name created by the original Helm chart. `dataSource.adhoc.backupName` is the backup name of the data to be restored.
  3. Create a new `Job` using the modified `values.yaml` file and resume data restore.

4. After the troubleshooting and data restore is completed, delete the `Jobs` for data restore and troubleshooting.

9 Troubleshoot

9.1 Tips for troubleshooting TiDB in Kubernetes

This document describes the commonly used tips for troubleshooting TiDB in Kubernetes.

9.1.1 Use the diagnostic mode

When a Pod is in the `CrashLoopBackoff` state, the containers in the Pod exit continually. As a result, you cannot use `kubectl exec` normally, making it inconvenient to diagnose issues.

To solve this problem, TiDB Operator provides the Pod diagnostic mode for PD, TiKV, and TiDB components. In this mode, the containers in the Pod hang directly after they are started, and will not repeatedly crash. Then you can use `kubectl exec` 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
```

When the container in the Pod is restarted again, it will detect this annotation and enter the diagnostic mode.

2. Wait for the Pod to enter the Running state.

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

Here's an example of using `kubectl exec` to get into the container for diagnosis:

```
kubectl exec -it ${pod_name} -n ${namespace} -- /bin/sh
```

3. 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.

9.2 Common Deployment Failures of TiDB in Kubernetes

This document describes the common deployment failures of TiDB in Kubernetes and their solutions.

9.2.1 The Pod is not created normally

After creating a cluster, if the Pod is not created, you can diagnose it using the following commands:

```
kubectl get tidbclusters -n ${namespace} && \
kubectl describe tidbclusters -n ${namespace} ${cluster_name} && \
kubectl get statefulsets -n ${namespace} && \
kubectl describe statefulsets -n ${namespace} ${cluster_name}-pd
```

After creating a backup/restore task, if the Pod is not created, you can perform a diagnostic operation by executing the following commands:

```
kubectl get backups -n ${namespace}
kubectl get jobs -n ${namespace}
kubectl describe backups -n ${namespace} ${backup_name}
kubectl describe backupschedules -n ${namespace} ${backupschedule_name}
kubectl describe jobs -n ${namespace} ${backupjob_name}
kubectl describe restores -n ${namespace} ${restore_name}
```

9.2.2 The Pod is in the Pending state

The Pending state of a Pod is usually caused by conditions of insufficient resources, for example:

- The `StorageClass` of the PVC used by PD, TiKV, TiFlash, Pump, Monitor, Backup, and Restore Pods 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}
```

9.2.2.1 CPU or memory resources are insufficient

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.

9.2.2.2 StorageClass of the PVC does not exist

If the `StorageClass` of the PVC cannot be found, take the following steps:

1. Get the available `StorageClass` in the cluster:

   ```
   kubectl get storageclass
   ```

2. Change `storageClassName` to the name of the `StorageClass` available in the cluster.

3. Update the configuration file:

   - If you want to start the TiDB cluster, execute `kubectl edit tc ${cluster_name } -n ${namespace}` to update the cluster.
   - If you want to run a backup/restore task, first execute `kubectl delete bk ${ backup_name} -n ${namespace}` to delete the old backup/restore task, and then execute `kubectl apply -f backup.yaml` to create a new backup/restore task.

4. Delete Statefulset and the corresponding PVCs:

   ```
   kubectl delete pvc -n ${namespace} ${pvc_name} && \
   kubectl delete sts -n ${namespace} ${statefulset_name}
   ```

9.2.2.3 Insufficient available PVs

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.

9.2.3 The high availability scheduling policy of tidb-scheduler is not satisfied

tidb-scheduler has a high availability scheduling policy for PD and TiKV. 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.

9.2.4 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`.

9.2.4.1 View the log of the current container

```
kubectl -n ${namespace} logs -f ${pod_name}
```

9.2.4.2 View the log when the container was last restarted

```
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.

9.2.4.3 "cluster id mismatch"

When the "cluster id mismatch" message appears in the TiKV Pod log, 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, this error might occur.

If you confirm 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.

9.2.4.4 `ulimit` is not big enough

TiKV might fail to start when `ulimit` is not big enough. 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
```

9.2.4.5 Other causes

If you cannot confirm the cause from the log and `ulimit` is also a normal value, troubleshoot the issue by using the diagnostic mode.

9.3 Common Cluster Exceptions of TiDB in Kubernetes

This document describes the common exceptions during the operation of TiDB clusters in Kubernetes and their solutions.

9.3.1 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 the TiKV component 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 ${cluster_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 cluster. Then the recovery is completed.

9.3.2 Persistent connections are abnormally terminated in TiDB

Load balancers often set the idle connection timeout. If no data is sent via a connection for a specific period of time, the load balancer closes the connection.

- If a persistent connection is terminated 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 versions.

9.4 Common Network Issues of TiDB in Kubernetes

This document describes the common network issues of TiDB in Kubernetes and their solutions.

9.4.1 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 among Pods from the log or monitoring metrics, or when you find the network connection among Pods might be abnormal according to the problematic condition, 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 ${cluster_name}-pd
kubectl -n ${namespace} get endpoints ${cluster_name}-tidb
kubectl -n ${namespace} get endpoints ${cluster_name}-pd-peer
kubectl -n ${namespace} get endpoints ${cluster_name}-tikv-peer
kubectl -n ${namespace} get endpoints ${cluster_name}-tidb-peer
```

The `ENDPOINTS` field shown in the above command must 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 Pod IP 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 port of the application 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
```

9.4.2 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 whether the TiDB service correctly generates the endpoint object:

```
kubectl get endpoints -n ${namespaces} ${cluster_name}-tidb
```

3. Check the log of TiDB components to see whether errors are reported.

```
kubectl logs -f ${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 `clusterIP` on the node. If the `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 `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 `clusterIP` and `PodIP`. Check the following items:

    - 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:

```
kubectl get endpoints -n ${namespaces} ${cluster_name}-tidb
```

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.

10 TiDB FAQs in Kubernetes

This document collects frequently asked questions (FAQs) about the TiDB cluster in Kubernetes.

10.1 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:

10.1.1 For the first deployment

Configure the `.spec.timezone` attribute in the TidbCluster CR. For example:

```
...
spec:
  timezone: Asia/Shanghai
...
```

Then deploy the TiDB cluster.

10.1.2 For a running cluster

If the TiDB cluster is already running, first upgrade the cluster, and then configure it to support the new time zone.

1. Upgrade the TiDB cluster:

   Configure the `.spec.timezone` attribute in the TidbCluster CR. For example:

```
...
spec:
  timezone: Asia/Shanghai
...
```

Then upgrade the TiDB cluster.

2. Configure TiDB to support the new time zone:

   Refer to Time Zone Support to modify TiDB service time zone settings.

10.2 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.

10.3 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.

10.4 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.

10.5 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.

10.6 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 ${pod_name} -n ${namespace} -- cat /etc/pd/pd.toml
```

- Check the TiKV configuration file:

```
kubectl exec -it ${pod_name} -n ${namespace} -- cat /etc/tikv/tikv.toml
```

- Check the TiDB configuration file:

```
kubectl exec -it ${pod_name} -c tidb -n ${namespace} -- cat /etc/tidb/
    ↪ tidb.toml
```

10.7 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 Deployment Failures 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 cases, 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
```

10.8 How does TiDB ensure data safety and reliability?

To ensure persistent storage of data, TiDB clusters deployed by TiDB Operator use Persistent Volume provided by Kubernetes cluster as the storage.

To ensure data safety in case one node is down, PD and TiKV use Raft Consistency Algorithm to replicate the stored data as multiple replicas across nodes.

In the bottom layer, TiKV replicates data using the log replication and State Machine model. For write requests, data is written to the Leader node first, and then the Leader node replicates the command to its Follower nodes as a log. When most of the Follower nodes in the cluster receive this log from the Leader node, the log is committed and the State Machine changes accordingly.

11 Reference

11.1 Architecture

11.1.1 TiDB Operator Architecture

This document describes the architecture of TiDB Operator and how it works.

11.1.1.1 Architecture

The following diagram is an overview of the architecture of TiDB Operator.

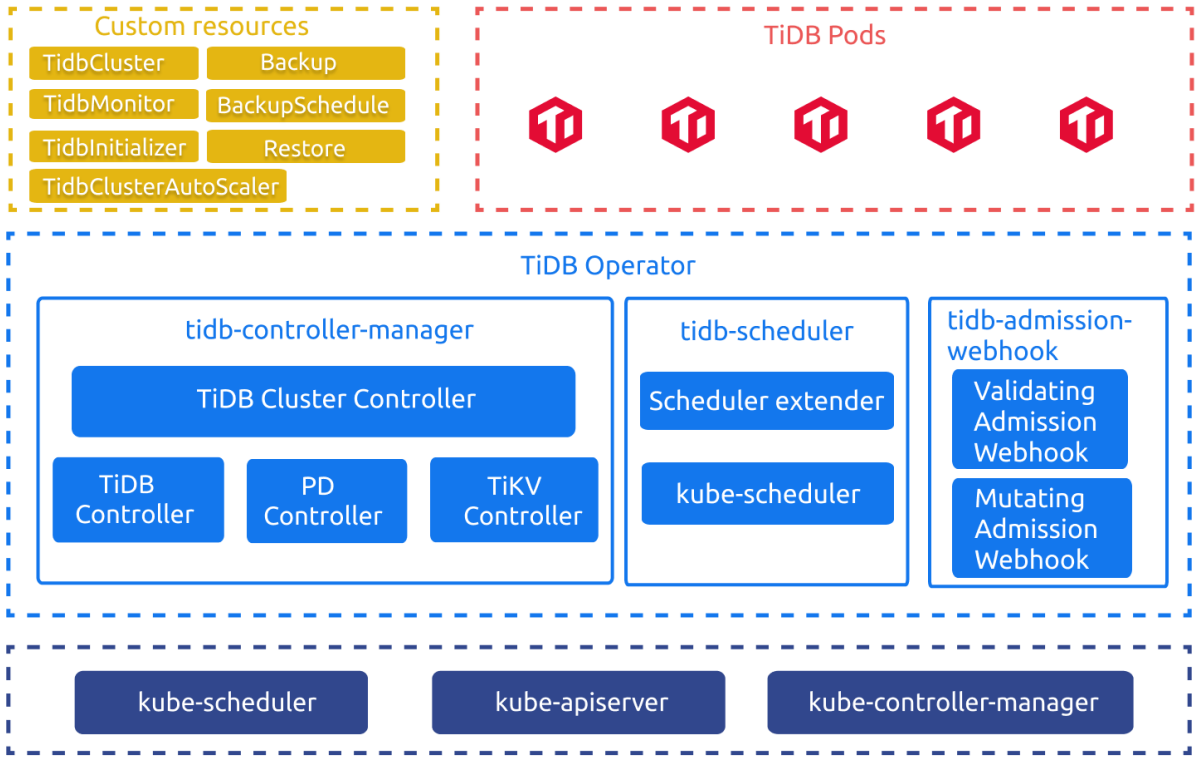


Figure 1: TiDB Operator Overview

`TidbCluster`, `TidbMonitor`, `TidbInitializer`, `Backup`, `Restore`, `BackupSchedule`, and `TidbClusterAutoScaler` are custom resources defined by CRD (`CustomResourceDefinition` ↪).

- `TidbCluster` describes the desired state of the TiDB cluster.
- `TidbMonitor` describes the monitoring components of the TiDB cluster.
- `TidbInitializer` describes the desired initialization Job of the TiDB cluster.
- `Backup` describes the desired backup of the TiDB cluster.
- `Restore` describes the desired restoration of the TiDB cluster.
- `BackupSchedule` describes the scheduled backup of the TiDB cluster.
- `TidbClusterAutoScaler` describes the automatic scaling 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 and complete the corresponding control logic according to other CRs;

- `tidb-scheduler` is a Kubernetes scheduler extension that injects the TiDB specific scheduling policies to the Kubernetes scheduler;
- `tidb-admission-webhook` is a dynamic admission controller in Kubernetes, which completes the modification, verification, operation, and maintenance of Pod, StatefulSet, and other related resources.

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

11.1.1.2 Control flow

The following diagram is the analysis of the control flow of TiDB Operator. Starting from TiDB Operator v1.1, the TiDB cluster, monitoring, initialization, backup, and other components are deployed and managed using CR.

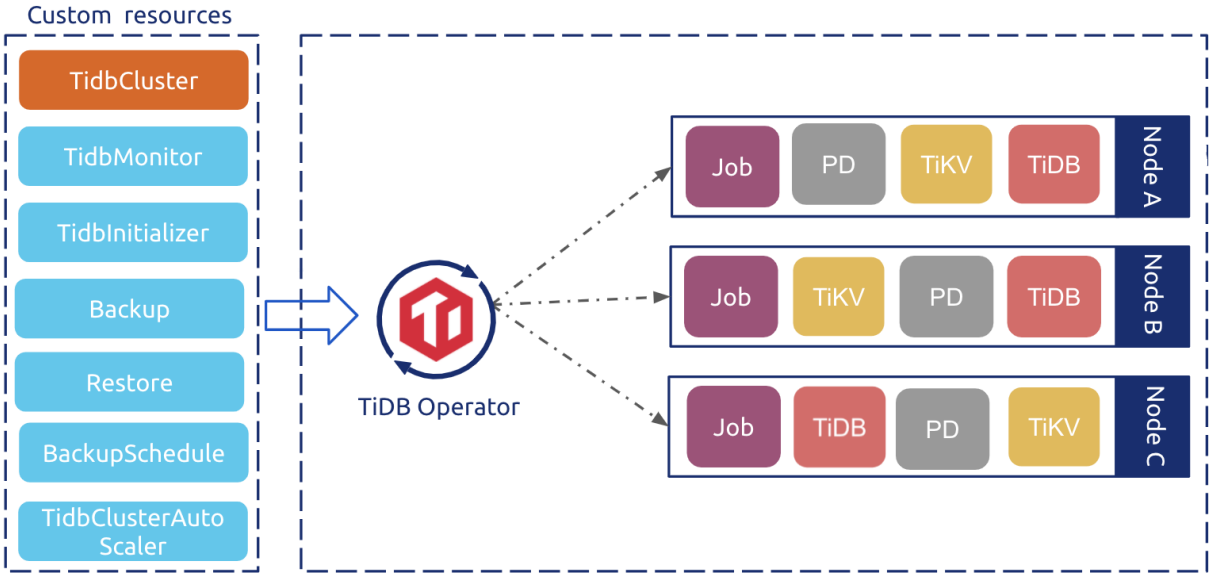


Figure 2: TiDB Operator Control Flow

The overall control flow is described as follows:

1. The user creates a `TidbCluster` object and other CR objects through kubectl, such as `TidbMonitor`;
2. TiDB Operator watches `TidbCluster` and other related objects, and constantly adjust the `StatefulSet`, `Deployment`, `Service`, and other objects of PD, TiKV, TiDB, Monitor or other components based on the actual state of the cluster;
3. Kubernetes' native controllers create, update, or delete the corresponding `Pod` based on objects such as `StatefulSet`, `Deployment`, and `Job`;

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.

11.1.2 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.

11.1.2.1 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.

Currently, pods can be scheduled according to specific dimensions by modifying `metadata.annotations` in TidbCluster, such as:

```
metadata:
  annotations:
    pingcap.com/ha-topology-key: kubernetes.io/hostname
```

Or by modifying tidb-cluster Chart `values.yaml`:

```
haTopologyKey: kubernetes.io/hostname
```

The configuration above indicates scheduling by the node dimension (default). If you want to schedule pods by other dimensions, such as `pingcap.com/ha-topology-key: zone`, which means scheduling by zone, each node should also be labeled as follows:

```
kubectl label nodes node1 zone=zone1
```

Different nodes may have different labels or the same label, and if a node is not labeled, the scheduler will not schedule any pod to that node.

TiDB Scheduler implements the following customized scheduling rules. The following example is based on node scheduling, scheduling rules based on other dimensions are the same.

11.1.2.1.1 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
…	

11.1.2.1.2 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
…		

11.1.2.1.3 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.

11.1.2.2 How TiDB Scheduler works

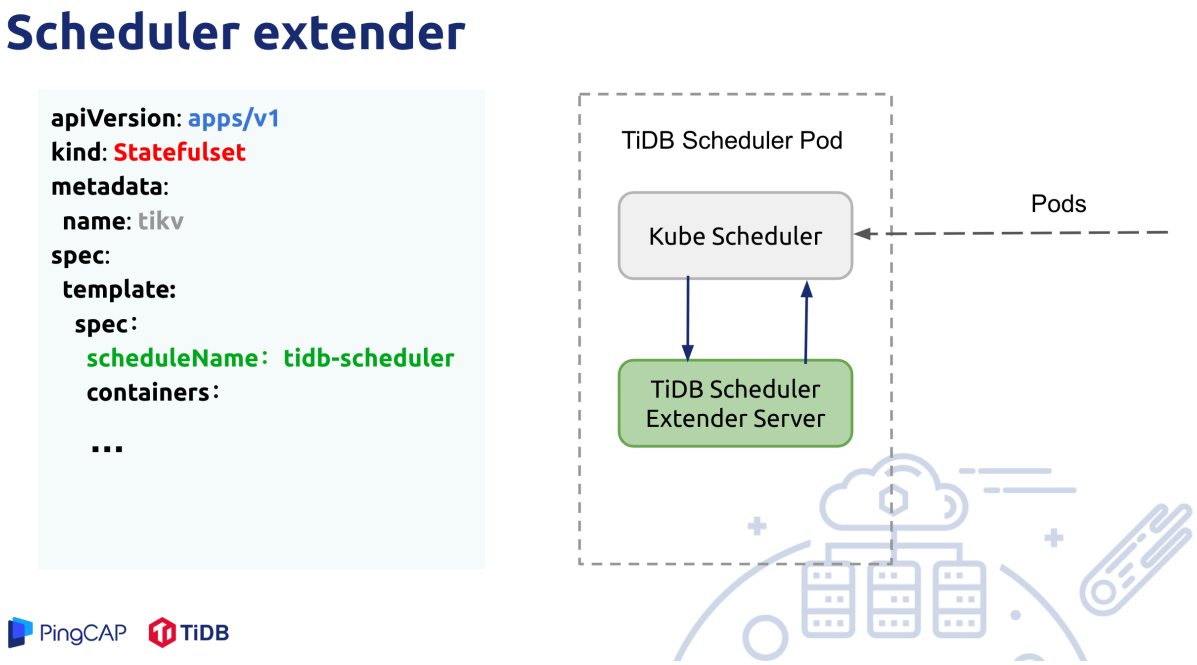


Figure 3: TiDB Scheduler Overview

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.

11.1.3 Advanced StatefulSet Controller

Feature Stage: Alpha

Kubernetes has a built-in StatefulSet that allocates consecutive serial numbers to Pods. For example, when there are three replicas, the Pods are named as pod-0, pod-1, and pod-2. When scaling out or scaling in, you must add a Pod at the end or delete the last pod. For example, when you scale out to four replicas, pod-3 is added. When you scale in to two replicas, pod-2 is deleted.

When you use local storage, Pods are associated with the Nodes storage resources and cannot be scheduled freely. If you want to delete one of the Pods in the middle to maintain its Node but no other Nodes can be migrated, or if you want to delete a Pod that fails and to create another Pod with a different serial number, you cannot implement such desired function by the built-in StatefulSet.

The advanced StatefulSet controller is implemented based on the built-in StatefulSet controller. It supports freely controlling the serial number of Pods. This document describes how to use the advanced StatefulSet controller in TiDB Operator.

11.1.3.1 Enable

1. Load the Advanced StatefulSet CRD file:

    - For Kubernetes versions < 1.16:

    ```
    kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
        ↪ operator/v1.1.15/manifests/advanced-statefulset-crd.v1beta1.
        ↪ yaml
    ```

    - For Kubernetes versions >= 1.16:

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

2. Enable the `AdvancedStatefulSet` feature in `values.yaml` of the TiDB Operator chart:

```
features:
- AdvancedStatefulSet=true
advancedStatefulset:
  create: true
```

3. Upgrade TiDB Operator. For details, refer to Upgrade TiDB Operator.

4. After upgrading TiDB Operator, check the AdvancedStatefulSet Controller is deployed by the following command:

```
kubectl get pods -n ${operator-ns} --selector app.kubernetes.io/
    ↪ component=advanced-statefulset-controller
```

Expected output

```
NAME                                             READY      STATUS
    ↪ RESTARTS  AGE
advanced-statefulset-controller-67885c5dd9-f522h 1/1    Running  0
    ↪          10s
```

Note:

If the `AdvancedStatefulSet` feature is enabled, TiDB Operator converts the current `StatefulSet` object into an `AdvancedStatefulSet` object. However, after the `AdvancedStatefulSet` feature is disabled, the `AdvancedStatefulSet` object cannot be automatically converted to the built-in `StatefulSet` object of Kubernetes.

11.1.3.2 Usage

This section describes how to use the advanced StatefulSet controller.

11.1.3.2.1 View the `AdvancedStatefulSet` Object by kubectl

The data format of `AdvancedStatefulSet` is the same as that of `StatefulSet`, but `AdvancedStatefulSet` is implemented based on CRD, with `asts` as the alias. You can view the `AdvancedStatefulSet` object in the namespace by running the following command:

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

11.1.3.2.2 Specify the Pod to be scaled in

With the advanced StatefulSet controller, when scaling in TidbCluster, you can not only reduce the number of replicas, but also specify the scaling in of any Pod in the PD, TiDB, or TiKV components by configuring annotations.

For example:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: asts
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 4
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
    config: {}
```

The above configuration deploys 4 TiKV instances, namely `basic-tikv-0`, `basic-tikv-1`, ..., `basic-tikv-3`. If you want to delete `basic-tikv-1`, set `spec.tikv.replicas` to 3 and configure the following annotations:

```
metadata:
  annotations:
    tikv.tidb.pingcap.com/delete-slots: '[1]'
```

Note:

> When modifying `replicas` and `delete-slots annotation`, complete the modification in the same operation; otherwise, the controller operates the modification according to the general expectations.

The complete example is as follows:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  annotations:
    tikv.tidb.pingcap.com/delete-slots: '[1]'
  name: asts
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
    config: {}
```

The supported annotations are as follows:

- `pd.tidb.pingcap.com/delete-slots`: Specifies the serial numbers of the Pods to be deleted in the PD component.
- `tidb.tidb.pingcap.com/delete-slots`: Specifies the serial number of the Pods to be deleted in the TiDB component.
- `tikv.tidb.pingcap.com/delete-slots`: Specifies the serial number of the Pods to be deleted in the TiKV component.

The value of Annotation is an integer array of JSON, such as `[0]`, `[0,1]`, `[1,3]`.

11.1.3.2.3 Specify the location to scale out

You can reverse the above operation of scaling in to restore `basic-tikv-1`.

> **Note:**
>
> The specified scaling out performed by the advanced StatefulSet controller is the same as the regular StatefulSet scaling, which does not delete the Persistent Volume Claims (PVCs) associated with the Pod. If you want to avoid using the previous data, delete the associated PVCs before scaling out at the original location.

For example:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  annotations:
    tikv.tidb.pingcap.com/delete-slots: '[]'
  name: asts
spec:
  version: v5.0.6
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    replicas: 4
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    replicas: 2
    service:
      type: ClusterIP
```

```
  config: {}
```

The delete-slots annotations can be left empty or deleted completely.

11.1.4 Enable Admission Controller in TiDB Operator

Kubernetes v1.9 introduces the dynamic admission control to modify and validate resources. TiDB Operator also supports the dynamic admission control to modify, validate, and maintain resources. This document describes how to enable the admission controller and introduces the functionality of the admission controller.

11.1.4.1 Prerequisites

Unlike those of most products on Kubernetes, the admission controller of TiDB Operator consists of two mechanisms: extension API-server and Webhook Configuration.

To use the admission controller, you need to enable the aggregation layer feature of the Kubernetes cluster. The feature is enabled by default. To check whether it is enabled, see Enable Kubernetes Apiserver flags.

11.1.4.2 Enable the admission controller

With a default installation, TiDB Operator disables the admission controller. Take the following steps to manually turn it on.

1. Edit the `values.yaml` file in TiDB Operator.

   Enable the `Operator Webhook` feature:

   ```
   admissionWebhook:
     create: true
   ```

   - If your Kubernetes cluster version >= v1.13.0, enable the Webhook feature by using the configuration above.
   - If your Kubernetes cluster version < v1.13.0, run the following command and configure the `admissionWebhook.cabundle` in `values.yaml` as the return value:

     ```
     kubectl get configmap -n kube-system extension-apiserver-
         ↪ authentication -o=jsonpath='{.data.client-ca-file}' | base64
         ↪  | tr -d '\n'
     ```

     ```
     admissionWebhook:
       # Configure the value of `admissionWebhook.cabundle` as the
           ↪ return value of the command above
       cabundle: <cabundle>
     ```

2. Configure the failure policy.

   Prior to Kubernetes v1.15, the management mechanism of the dynamic admission control is coarser-grained and is inconvenient to use. To prevent the impact of the dynamic admission control on the global cluster, you need to configure the Failure Policy.

   - For Kubernetes versions earlier than v1.15, it is recommended to set the `failurePolicy` of TiDB Operator to `Ignore`. This avoids the influence on the global cluster in case of `admission webhook` exception in TiDB Operator.

     ```
     ......
     failurePolicy:
         validation: Ignore
         mutation: Ignore
     ```

   - For Kubernetes v1.15 and later versions, it is recommended to set the `failurePolicy` of TiDB Operator to `Failure`. The exception occurs in `admission webhook` does not affect the whole cluster, because the dynamic admission control supports the label-based filtering mechanism.

     ```
     ......
     failurePolicy:
         validation: Fail
         mutation: Fail
     ```

3. Install or update TiDB Operator.

   To install or update TiDB Operator, see Deploy TiDB Operator in Kubernetes.

11.1.4.3 Set the TLS certificate for the admission controller

By default, the admission controller and Kubernetes api-server skip the TLS verification. To manually enable and configure the TLS verification between the admission controller and Kubernetes api-server, take the following steps:

1. Generate the custom certificate.

   To generate the custom CA (client auth) file, refer to Step 1 to Step 4 in Generate certificates using `cfssl`.

   Use the following configuration in `ca-config.json`:

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

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

After executing Step 4, run the `ls` command. The following files should be listed in the `cfssl` folder:

```
ca-config.json  ca-csr.json  ca-key.pem  ca.csr  ca.pem
```

2. Generate the certificate for the admission controller.

    1. Create the default `webhook-server.json` file:

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

    2. Modify the `webhook-server.json` file as follows:

    ```
    {
        "CN": "TiDB Operator Webhook",
        "hosts": [
            "tidb-admission-webhook.<namespace>",
            "tidb-admission-webhook.<namespace>.svc",
            "tidb-admission-webhook.<namespace>.svc.cluster",
            "tidb-admission-webhook.<namespace>.svc.cluster.local"
        ],
        "key": {
            "algo": "rsa",
            "size": 2048
        },
        "names": [
            {
                "C": "US",
                "L": "CA",
                "O": "PingCAP",
                "ST": "Beijing",
                "OU": "TiDB"
            }
        ]
    }
    ```

`<namespace>` is the namespace which TiDB Operator is deployed in.

3. Generate the server-side certificate for TiDB Operator Webhook:

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

4. Run the `ls | grep webhook-server` command. The following files should be listed:

```
webhook-server-key.pem
webhook-server.csr
webhook-server.json
webhook-server.pem
```

3. Create a secret in the Kubernetes cluster:

```
kubectl create secret generic <secret-name> --namespace=<namespace> --
    ↪ from-file=tls.crt=~/cfssl/webhook-server.pem --from-file=tls.key
    ↪ =~/cfssl/webhook-server-key.pem --from-file=ca.crt=~/cfssl/ca.pem
```

4. Modify `values.yaml`, and install or upgrade TiDB Operator.

   Get the value of `ca.crt`:

```
kubectl get secret <secret-name> --namespace=<release-namespace> -o=
    ↪ jsonpath='{.data.ca\.crt}'
```

   Configure the items in `values.yaml` as described below:

```
admissionWebhook:
  apiservice:
    insecureSkipTLSVerify: false # Enable TLS verification
    tlsSecret: "<secret-name>" # The name of the secret created in Step
        ↪  3
    caBundle: "<caBundle>" # The value of `ca.crt` obtained in the
        ↪ above step
```

   After configuring the items, install or upgrade TiDB Operator. For installation, see Deploy TiDB Operator. For upgrade, see Upgrade TiDB Operator.

11.1.4.4 Functionality of the admission controller

TiDB Operator implements many functions using the admission controller. This section introduces the admission controller for each resource and its corresponding functions.

- Admission controller for Pod validation

  The admission controller for Pod validation guarantees the safe logon and safe logoff of the PD/TiKV/TiDB component. You can restart a TiDB cluster in Kubernetes using this controller. The component is enabled by default if the admission controller is enabled.

```
admissionWebhook:
  validation:
    pods: true
```

- Admission controller for StatefulSet validation

  The admission controller for StatefulSet validation supports the gated launch of the TiDB/TiKV component in a TiDB cluster. The component is disabled by default if the admission controller is enabled.

```
admissionWebhook:
  validation:
    statefulSets: false
```

  You can control the gated launch of the TiDB/TiKV component in a TiDB cluster through two annotations, `tidb.pingcap.com/tikv-partition` and `tidb.pingcap.com/tidb-partition`. To set the gated launch of the TiKV component in a TiDB cluster, execute the following commands. The effect of `partition=2` is the same as that of StatefulSet Partitions.

```
kubectl annotate tidbcluster ${name} -n ${namespace} tidb.pingcap.com/tikv-partition=2 &&
tidbcluster.pingcap.com/${name} annotated
```

  Execute the following commands to unset the gated launch:

```
kubectl annotate tidbcluster ${name} -n ${namespace} tidb.pingcap.com/tikv-partition- &&
tidbcluster.pingcap.com/${name} annotated
```

  This also applies to the TiDB component.

- Admission controller for TiDB Operator resources validation

  The admission controller for TiDB Operator resources validation supports validating customized resources such as `TidbCluster` and `TidbMonitor` in TiDB Operator. The component is disabled by default if the admission controller is enabled.

```
admissionWebhook:
  validation:
    pingcapResources: false
```

For example, regarding `TidbCluster` resources, the admission controller for TiDB Operator resources validation checks the required fields of the `spec` field. When you create or update `TidbCluster`, if the check is not passed (for example, neither of the `spec.pd.image` filed and the `spec.pd.baseImage` field is defined), this admission controller refuses the request.

- Admission controller for Pod modification

  The admission controller for Pod modification supports the hotspot scheduling of TiKV in the auto-scaling scenario. To enable TidbCluster auto-scaling, you need to enable this controller. The component is enabled by default if the admission controller is enabled.

```
admissionWebhook:
  mutation:
    pods: true
```

- Admission controller for TiDB Operator resources modification

  The admission controller for TiDB Operator resources modification supports filling in the default values of customized resources, such as `TidbCluster` and `TidbMonitor` in TiDB Operator. The component is enabled by default if the admission controller is enabled.

```
admissionWebhook:
  mutation:
    pingcapResources: true
```

11.2 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 affect the performance of TiDB in Kubernetes.

11.2.1 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

11.2.2 Test environment

11.2.2.1 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
```

11.2.2.2 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;
```

11.2.2.3 Hardware recommendations

11.2.2.3.1 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, the disk also performs differently. Therefore, only the machines in us-central1 are applied for test.

11.2.2.3.2 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]
```

11.2.2.3.3 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.

11.2.2.3.4 CPU

- In the test on a single AZ cluster, the c2-standard-16 machine mode is chosen for TiDB/TiKV.
- In the comparison test on a 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.

11.2.2.4 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`.

11.2.2.5 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.

11.2.2.5.1 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 the 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).

11.2.2.5.2 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
```

11.2.2.5.3 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}`.

11.2.3 Test report

11.2.3.1 In single AZ

11.2.3.1.1 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:

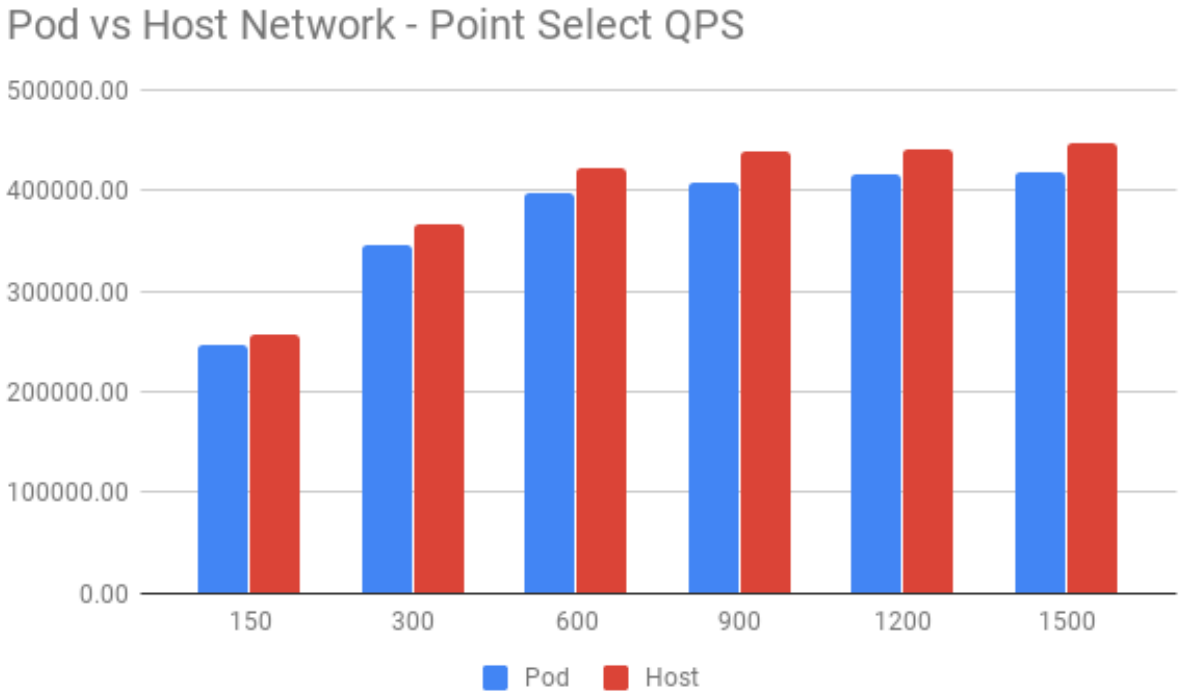


Figure 4: Pod vs Host Network

Latency comparison:

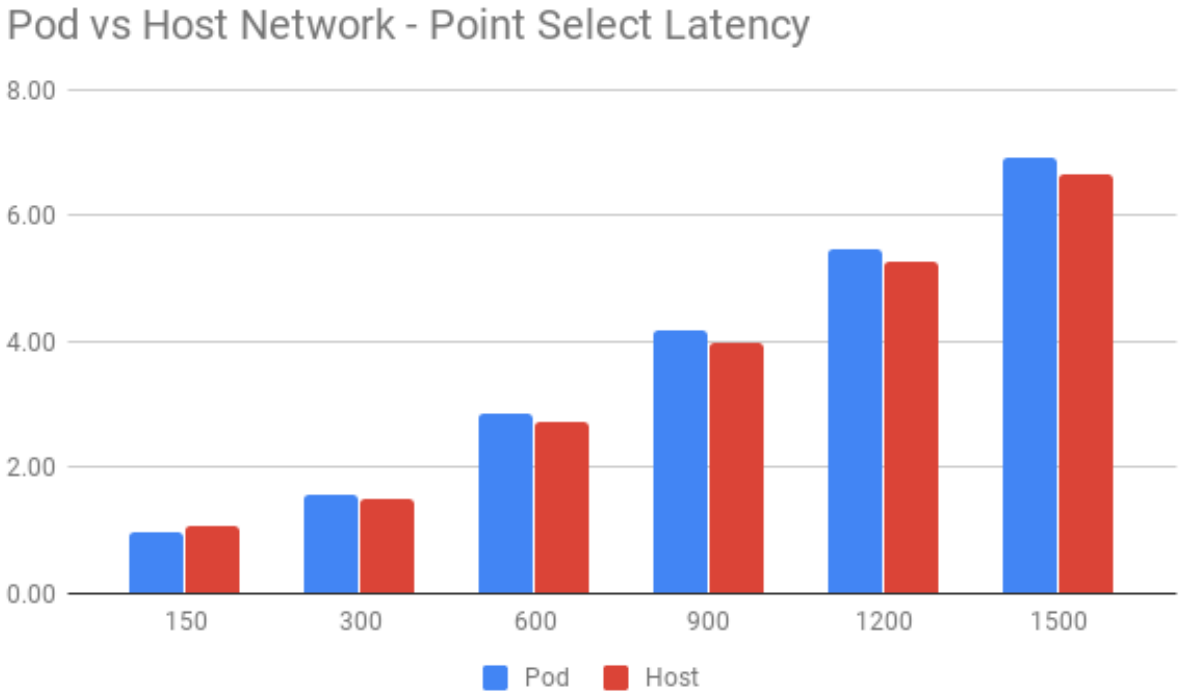


Figure 5: Pod vs Host Network

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

11.2.3.1.2 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:

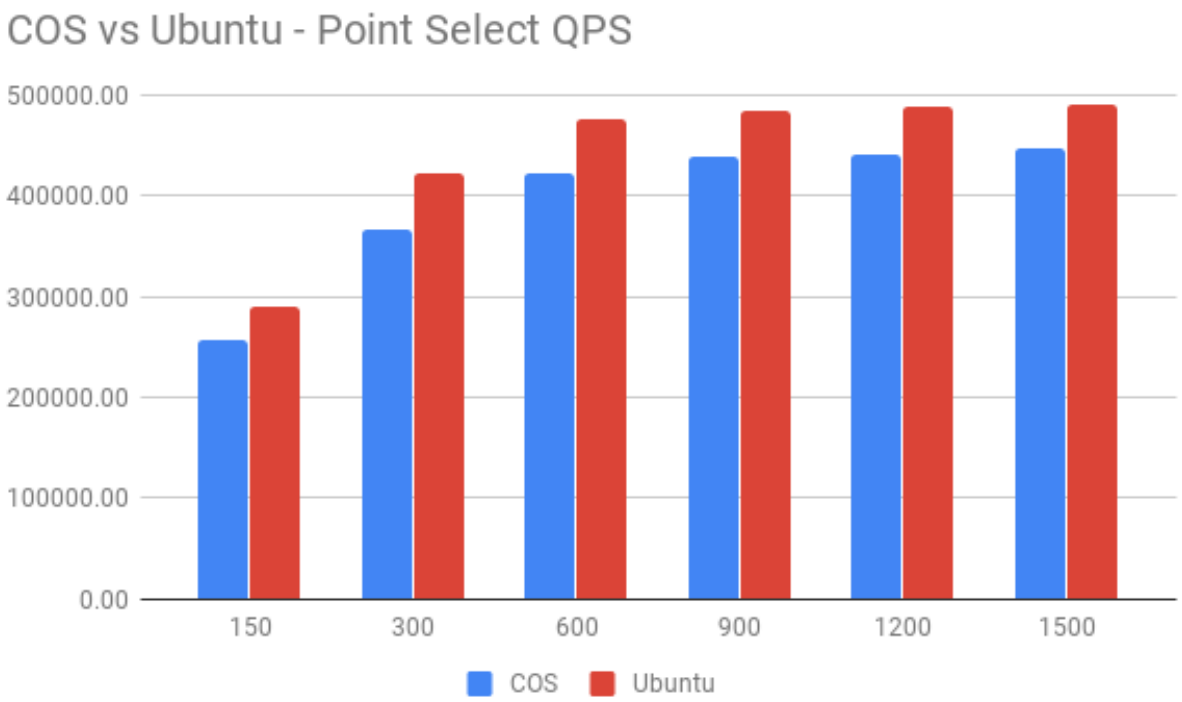


Figure 6: COS vs Ubuntu

Latency comparison:

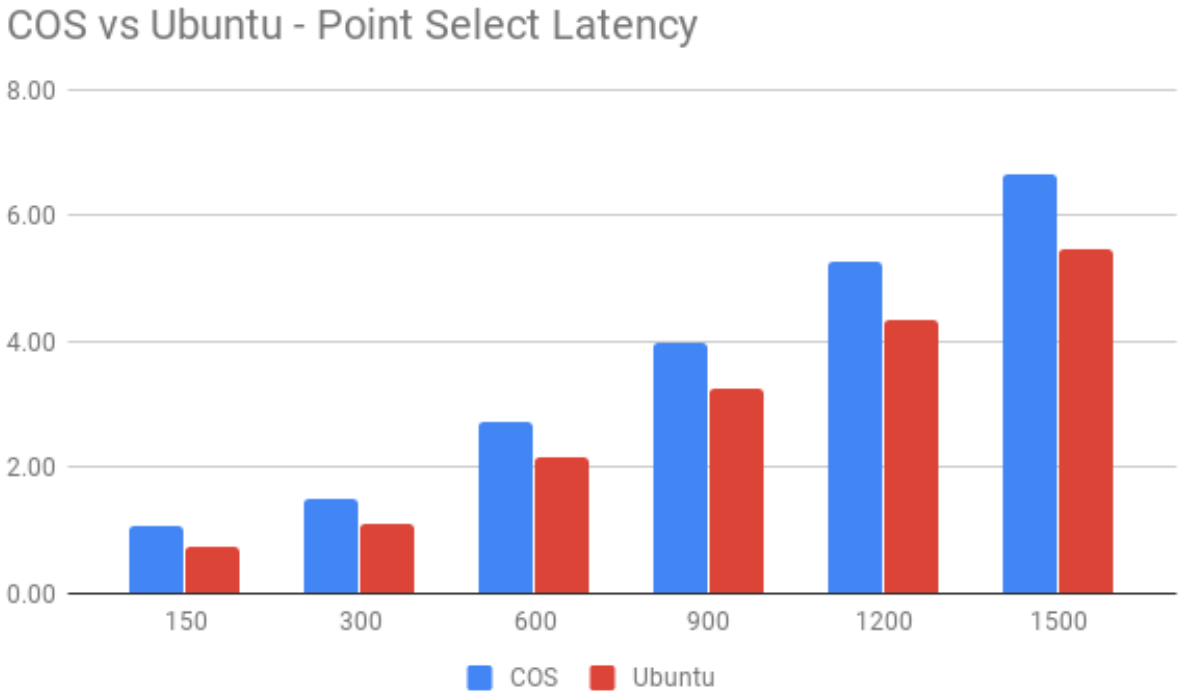


Figure 7: COS vs Ubuntu

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.

11.2.3.1.3 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:

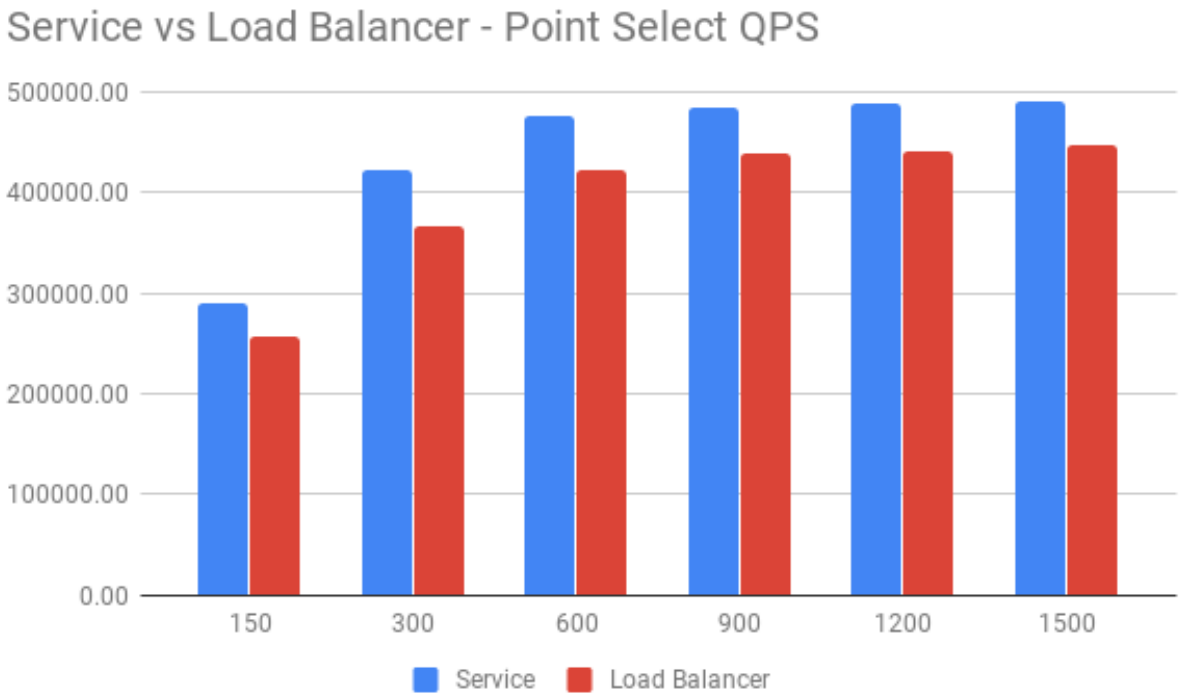


Figure 8: Service vs Load Balancer

Latency comparison:

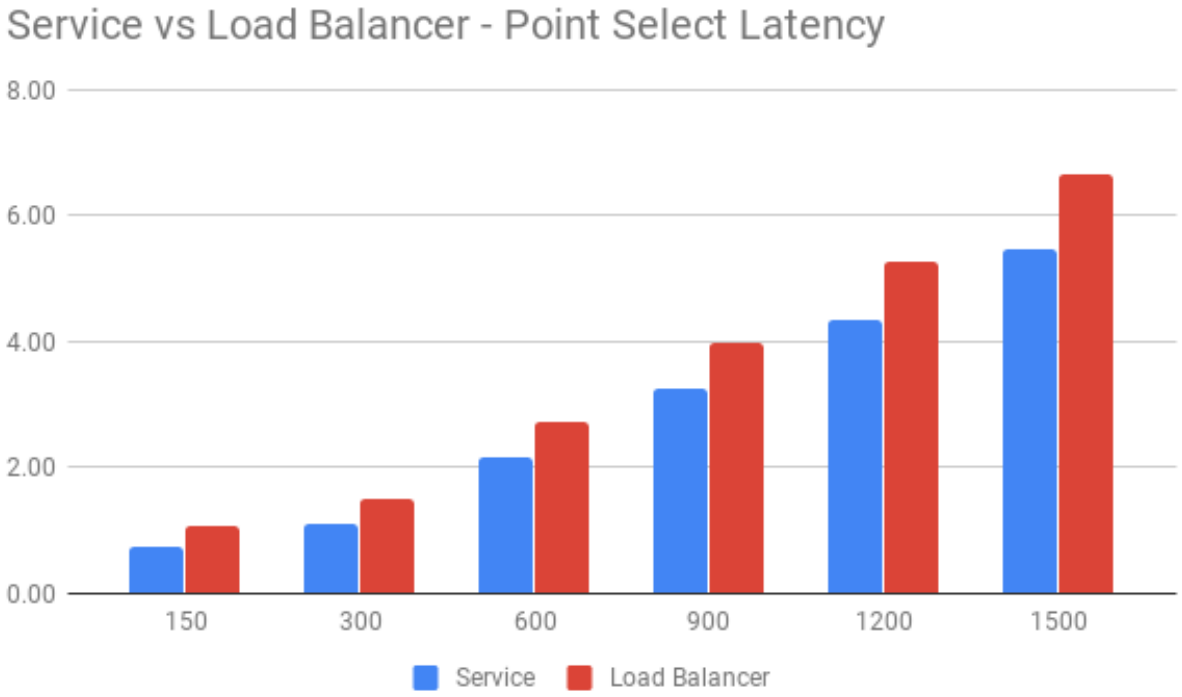


Figure 9: Service vs Load Balancer

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

11.2.3.1.4 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-standard-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:

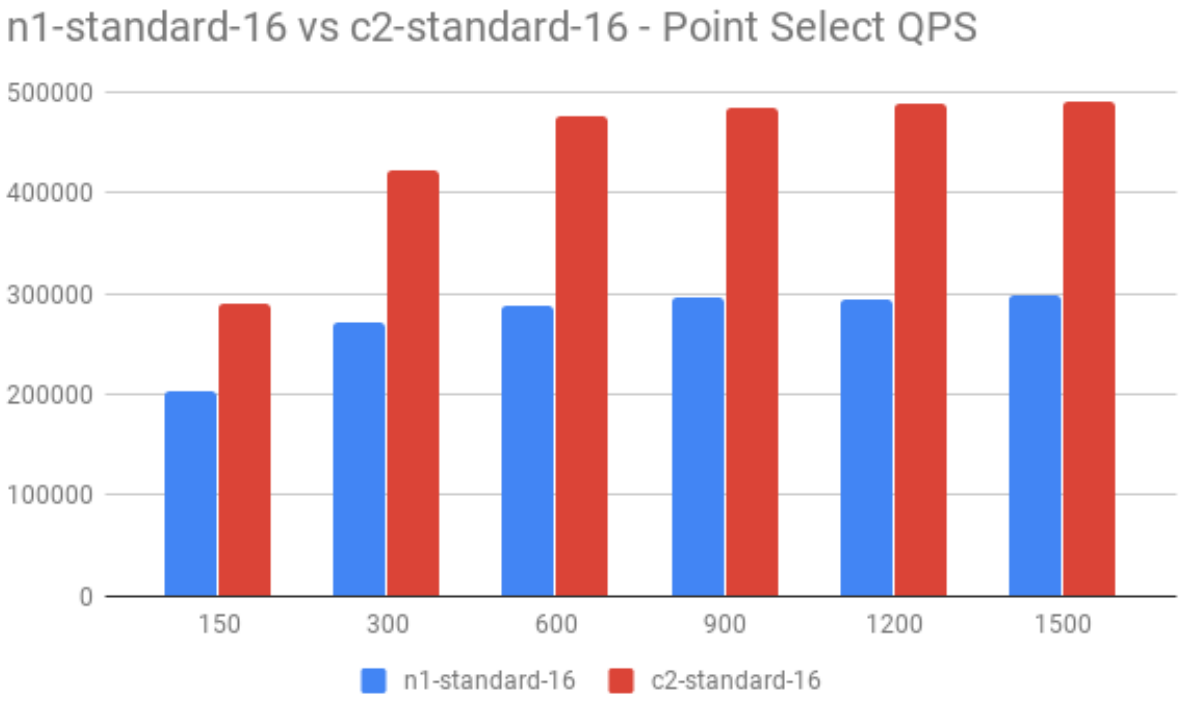


Figure 10: n1-standard-16 vs c2-standard-16

Latency comparison:

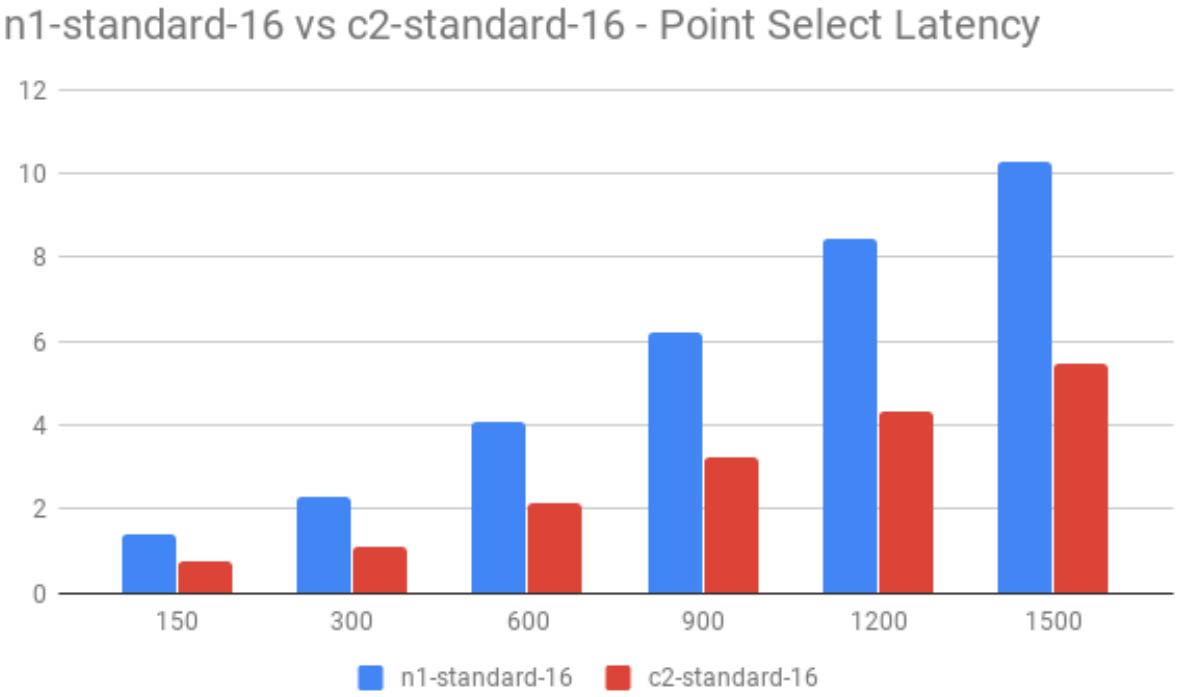


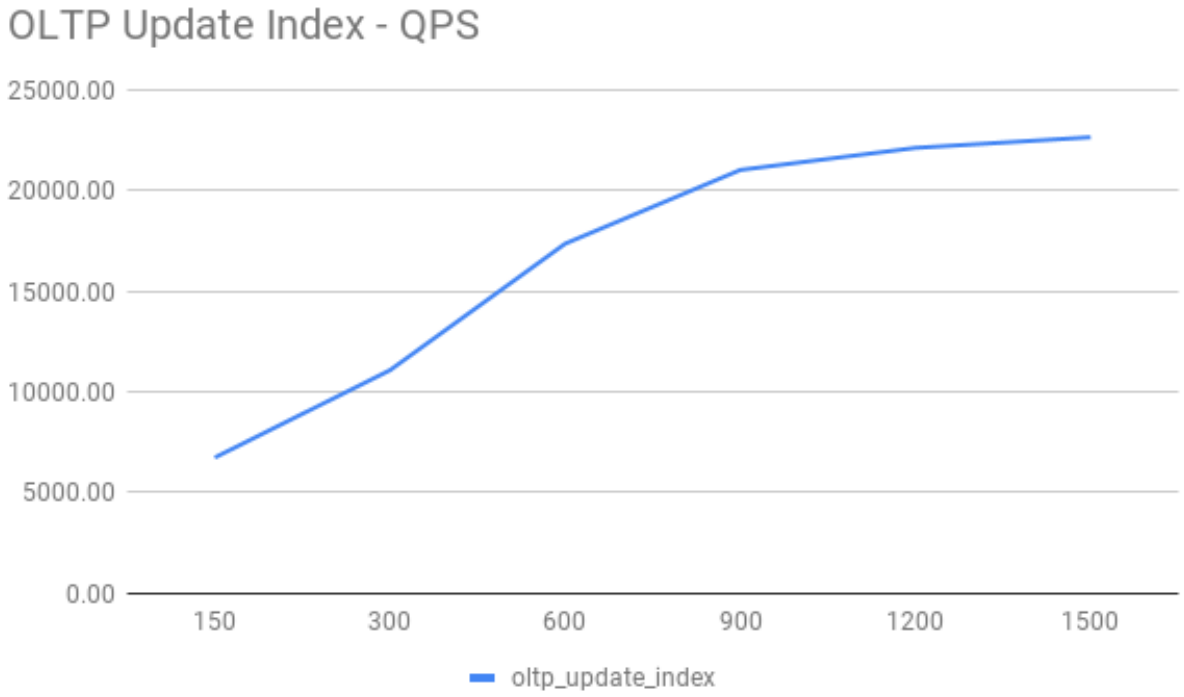
Figure 11: n1-standard-16 vs c2-standard-16

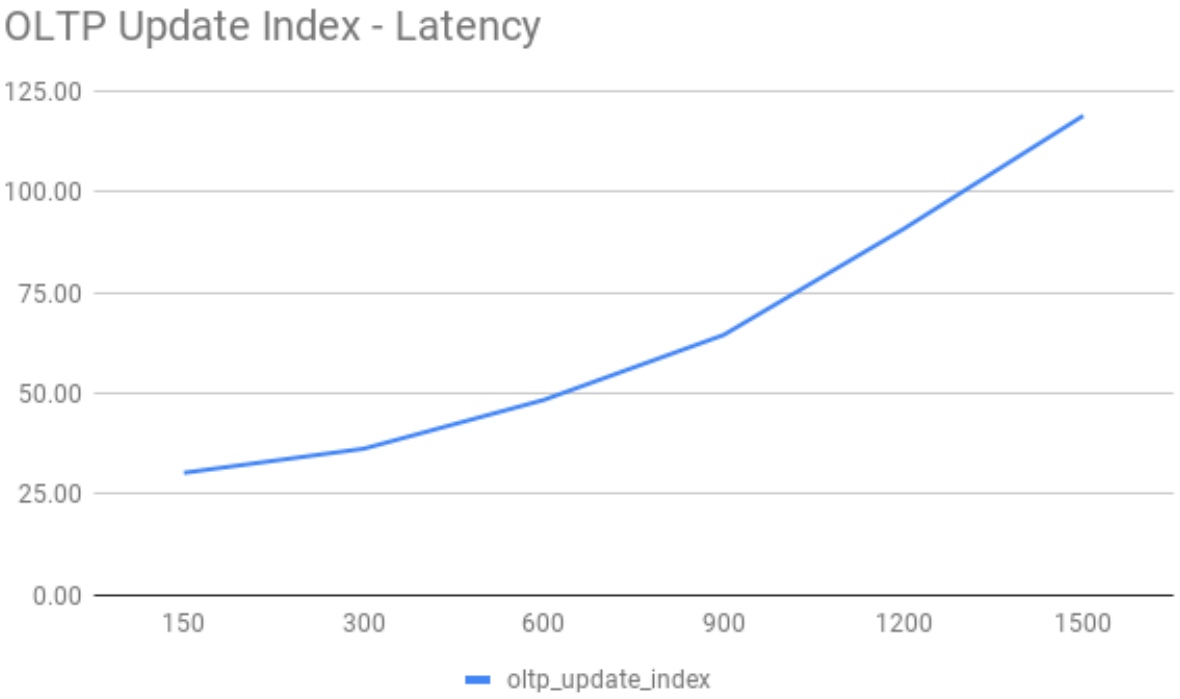
11.2.3.2 OLTP and other tests

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.

11.2.3.2.1 OLTP Update Index

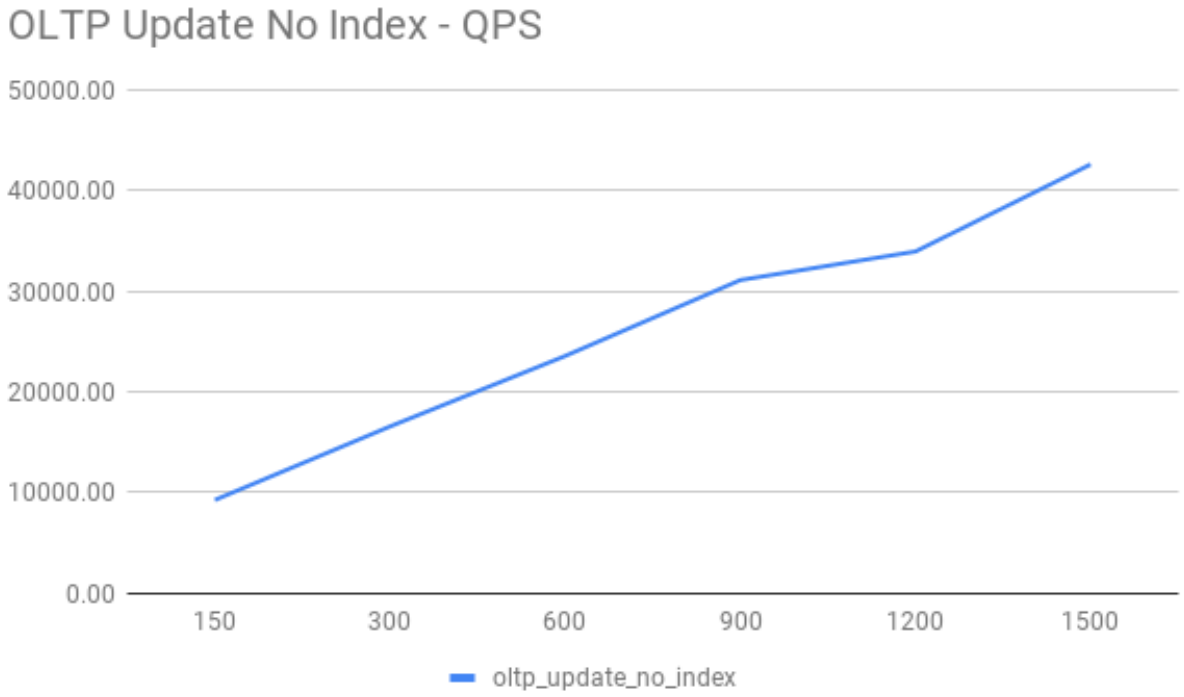
Threads	QPS	95% latency(ms)
150	6726.59	30.26
300	11067.55	36.24
600	17358.46	48.34
900	21025.23	64.47
1200	22121.87	90.78
1500	22650.13	118.92

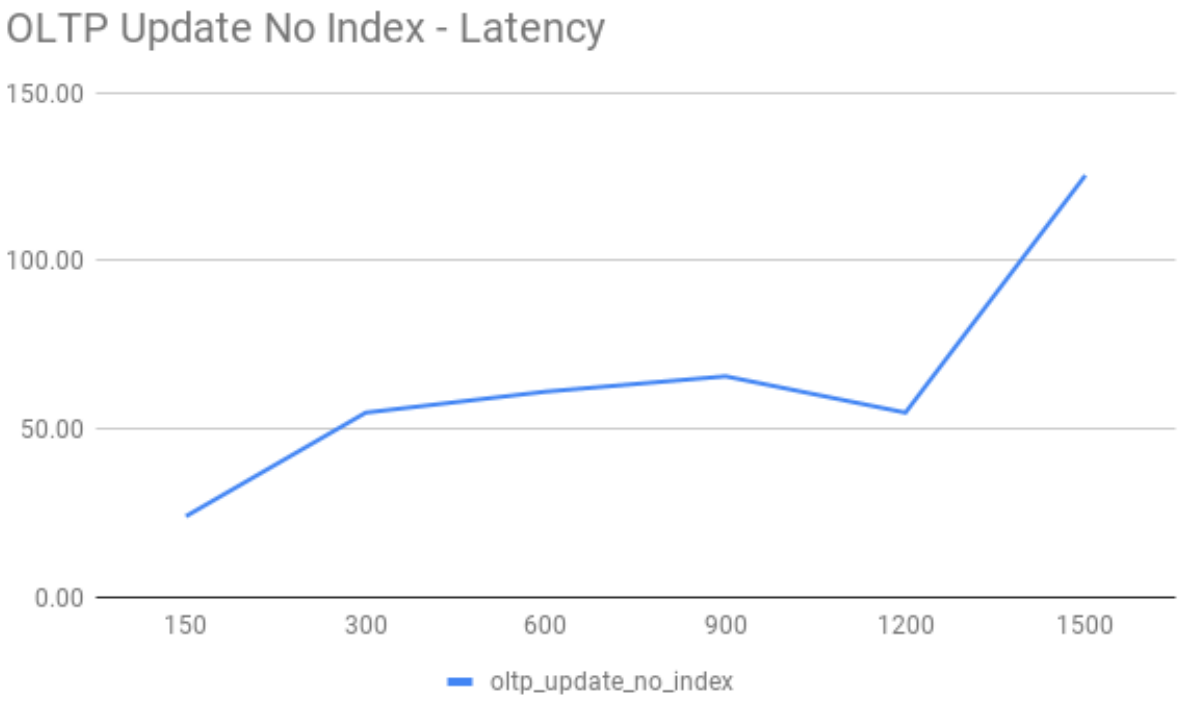




11.2.3.2.2 OLTP Update Non Index

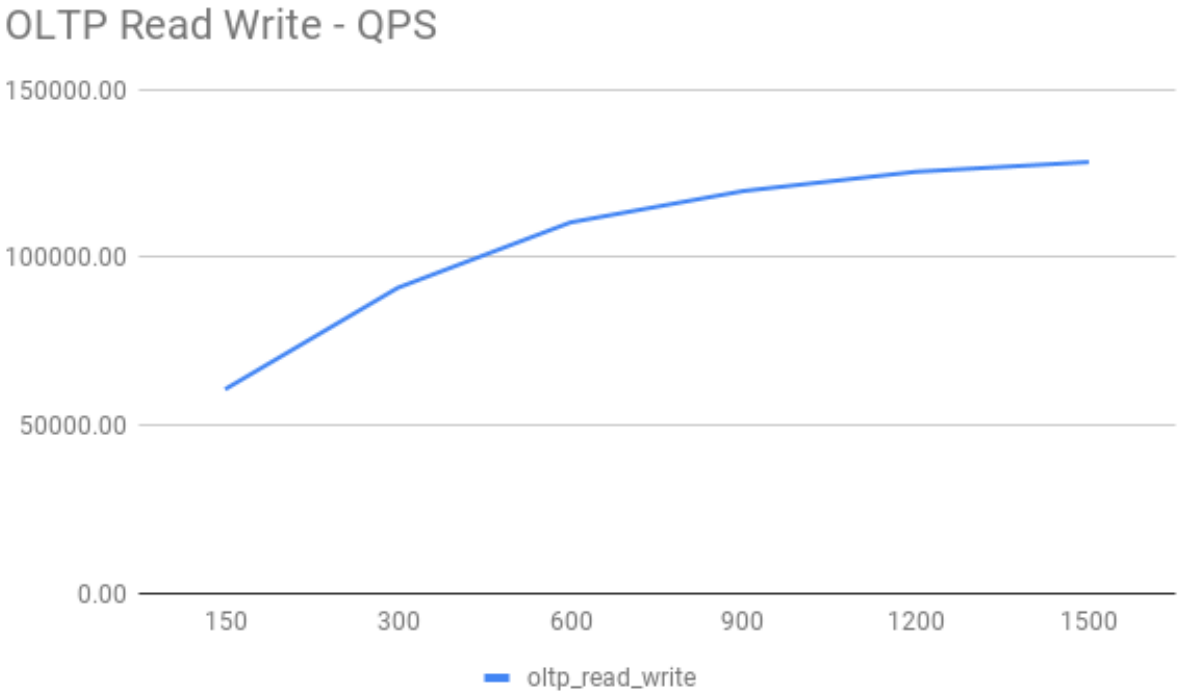
Threads	QPS	95% latency(ms)
150	9230.60	23.95
300	16543.63	54.83
600	23551.01	61.08
900	31100.10	65.65
1200	33942.60	54.83
1500	42603.13	125.52

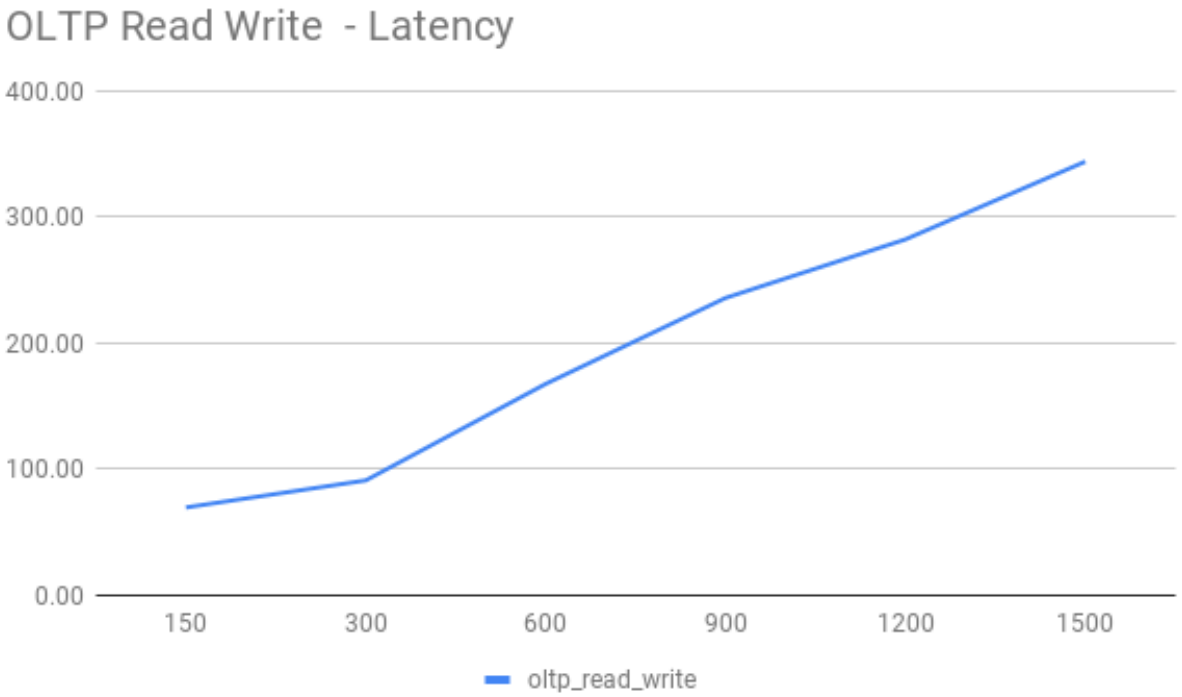




11.2.3.2.3 OLTP Read Write

Threads	QPS	95% latency(ms)
150	60732.84	69.29
300	91005.98	90.78
600	110517.67	167.44
900	119866.38	235.74
1200	125615.89	282.25
1500	128501.34	344.082





11.2.3.3 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:

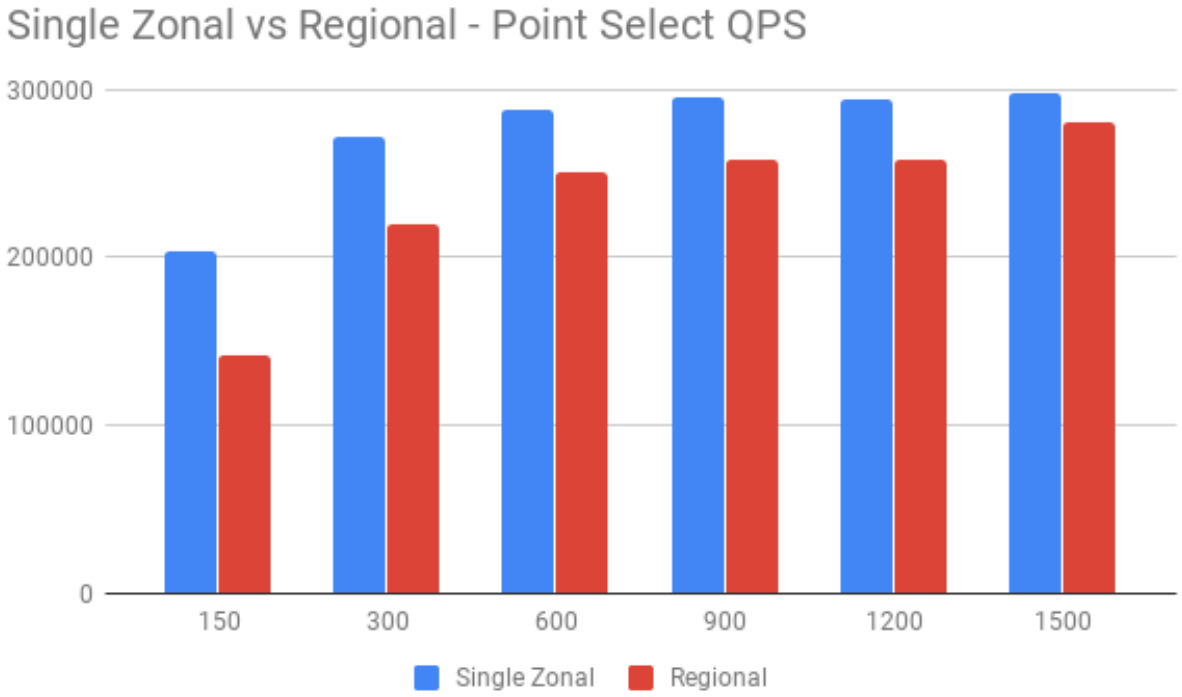


Figure 12: Single Zonal vs Regional

Latency comparison:

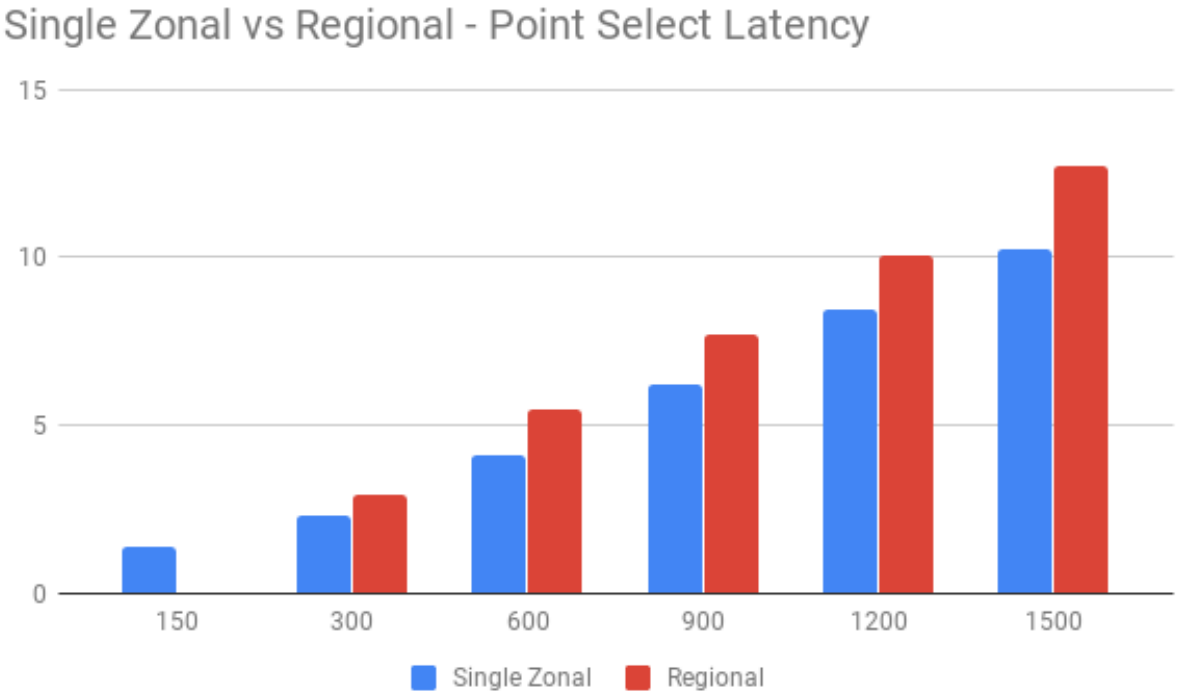


Figure 13: Single Zonal vs Regional

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.

11.2.4 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 vary 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, the difference between operating systems, the limit of Host network, and so on).

11.3 API References

11.4 Command Cheat Sheet for TiDB Cluster Management

This document is an overview of the commands used for TiDB cluster management.

11.4.1 kubectl

11.4.1.1 View resources

- View CRD:

```
kubectl get crd
```

- View TidbCluster:

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

- View TidbMonitor:

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

- View Backup:

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

- View BackupSchedule:

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

- View Restore:

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

- View TidbClusterAutoScaler:

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

- View TidbInitializer:

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

- View Advanced StatefulSet:

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

- View a Pod:

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

  View a TiKV Pod:

```
kubectl -n ${namespace} get pod -l app.kubernetes.io/component=tikv
```

  View the continuous status change of a Pod:

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

  View the detailed information of a Pod:

```
kubectl -n ${namespace} describe pod ${name}
```

- View the node on which Pods are located:

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

- View Service:

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

- View ConfigMap:

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

- View a PersistentVolume (PV):

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

View the PV used by the cluster:

```
kubectl get pv -l app.kubernetes.io/namespace=${namespace},app.
    ↪ kubernetes.io/managed-by=tidb-operator,app.kubernetes.io/instance
    ↪ =${cluster_name}
```

- View a PersistentVolumeClaim (PVC):

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

- View StorageClass:

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

- View StatefulSet:

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

View the detailed information of StatefulSet:

```
kubectl -n ${namespace} describe sts ${name}
```

11.4.1.2 Update resources

- Add an annotation for TiDBCluster:

```
kubectl -n ${namespace} annotate tc ${cluster_name} ${key}=${value}
```

Add a force-upgrade annotation for TiDBCluster:

```
kubectl -n ${namespace} annotate --overwrite tc ${cluster_name} tidb.
    ↪ pingcap.com/force-upgrade=true
```

Delete a force-upgrade annotation for TiDBCluster:

```
kubectl -n ${namespace} annotate tc ${cluster_name} tidb.pingcap.com/
    ↪ force-upgrade-
```

Enable the debug mode for Pods:

```
kubectl -n ${namespace} annotate pod ${pod_name} runmode=debug
```

11.4.1.3 Edit resources

- Edit TidbCluster:

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

11.4.1.4 Patch Resources

- Patch PV ReclaimPolicy:

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

- Patch a PVC:

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

- Patch StorageClass:

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

11.4.1.5 Create resources

- Create a cluster using the YAML file:

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

- Create Namespace:

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

- Create Secret:

  Create Secret of the certificate:

```
kubectl -n ${namespace} create secret generic ${secret_name} --from-
    ↪ file=tls.crt=${cert_path} --from-file=tls.key=${key_path} --from-
    ↪ file=ca.crt=${ca_path}
```

  Create Secret of the user id and password:

```
kubectl -n ${namespace} create secret generic ${secret_name} --from-
    ↪ literal=user=${user} --from-literal=password=${password}
```

11.4.1.6 Interact with running Pods

- View the PD configuration file:

```
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/pd/pd.toml
```

- View the TiDB configuration file:

```
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/tidb/tidb.toml
```

- View the TiKV configuration file:

```
kubectl -n ${namespace} -it exec ${pod_name} -- cat /etc/tikv/tikv.toml
```

- View Pod logs:

```
kubectl -n ${namespace} logs ${pod_name} -f
```

  View logs of the previous container:

```
kubectl -n ${namespace} logs ${pod_name} -p
```

  If there are multiple containers in a Pod, view logs of one container:

```
kubectl -n ${namespace} logs ${pod_name} -c ${container_name}
```

- Expose services:

```
kubectl -n ${namespace} port-forward svc/${service_name} ${local_port}:
    ↪ ${port_in_pod}
```

  Expose PD services:

```
kubectl -n ${namespace} port-forward svc/${cluster_name}-pd 2379:2379
```

11.4.1.7 Interact with nodes

- Mark the node as unschedulable:

```
kubectl cordon ${node_name}
```

- Mark the node as schedulable:

```
kubectl uncordon ${node_name}
```

11.4.1.8 Delete resources

- Delete a Pod:

```
kubectl delete -n ${namespace} pod ${pod_name}
```

- Delete a PVC:

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

- Delete TidbCluster:

```
kubectl delete -n ${namespace} tc ${tc_name}
```

- Delete TidbMonitor:

```
kubectl delete -n ${namespace} tidbmonitor ${tidb_monitor_name}
```

- Delete TidbClusterAutoScaler:

```
kubectl -n ${namespace} delete tidbclusterautoscaler ${name}
```

11.4.1.9 More

See kubectl Cheat Sheet for more kubectl usage.

11.4.2 Helm

11.4.2.1 Add Helm repository

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

11.4.2.2 Update Helm repository

```
helm repo update
```

11.4.2.3 View available Helm chart

- View charts in Helm Hub:

```
helm search hub ${chart_name}
```

For example:

```
helm search hub mysql
```

- View charts in other Repos:

```
helm search repo ${chart_name} -l --devel
```

For example:

```
helm search repo tidb-operator -l --devel
```

11.4.2.4 Get the default `values.yaml` of the Helm chart

```
helm inspect values ${chart_name} --version=${chart_version} > values.yaml
```

For example:

```
helm inspect values pingcap/tidb-operator --version=v1.1.15 > values-tidb-
    ↪ operator.yaml
```

11.4.2.5 Deploy using Helm chart

```
helm install ${name} ${chart_name} --namespace=${namespace} --version=${
    ↪ chart_version} -f ${values_file}
```

For example:

```
helm install tidb-operator pingcap/tidb-operator --namespace=tidb-admin --
    ↪ version=v1.1.15 -f values-tidb-operator.yaml
```

11.4.2.6 View the deployed Helm release

```
helm ls
```

11.4.2.7 Update Helm release

```
helm upgrade ${name} ${chart_name} --version=${chart_version} -f ${
    ↪ values_file}
```

For example:

```
helm upgrade tidb-operator pingcap/tidb-operator --version=v1.1.15 -f values
    ↪ -tidb-operator.yaml
```

11.4.2.8 Delete Helm release

```
helm uninstall ${name} -n ${namespace}
```

For example:

```
helm uninstall tidb-operator -n tidb-admin
```

11.4.2.9 More

See Helm Commands for more Helm usage.

11.5 Tools

11.5.1 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.

> **Note:**
>
> PingCAP is no longer maintaining `tkctl` from v1.1.x, some of the following functions may not be usable, please use the equivalent `kubectl` commands directly.

11.5.1.1 Installation

To install `tkctl`, you can download the pre-built binary or build `tkctl` from source.

11.5.1.1.1 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.

11.5.1.1.2 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
```

11.5.1.1.3 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
```

11.5.1.1.4 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.

11.5.1.2 Commands

11.5.1.2.1 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
```

11.5.1.2.2 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
```

11.5.1.2.3 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
```

11.5.1.2.4 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
```

11.5.1.2.5 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
```

11.5.1.2.6 tkctl debug [pod_name]

This command is used to diagnose the Pods in a TiDB cluster. It launches a debug launcher Pod which then starts a debug container using the specified docker image on the same host of the target Pod. The container has necessary troubleshooting tools and shares the same namespaces with the container in the target Pod, so you can 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`
–launcher-image		Specify the docker image for the debug launcher pod which is responsible for launching the debug container; default to `pingcap/debug-launcher:latest`

Flag	Abbreviation	Description
–container	-c	Select the container to be diagnosed; default to the first container of the target Pod
–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.

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
```

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.

11.5.1.2.7 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`
–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
```

11.5.1.2.8 tkctl help [command]

This command is used to print help messages of the sub commands.

For example:

```
tkctl help debug
```

11.5.1.2.9 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

11.5.2 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.

11.5.2.1 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
```

11.5.2.2 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} ${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 enter the target TiKV Pod and use `tikv-ctl` to perform the operation. The steps are as follows:

    1. Enter the Diagnostic mode:

    ```
    kubectl annotate pod ${pod_name} -n ${namespace} runmode=debug
    ```

    2. Stop the TiKV process:

    ```
    kubectl exec ${pod_name} -n ${namespace} -c tikv -- kill -s TERM 1
    ```

    3. Wait for the TiKV container to restart, and enter the container:

    ```
    kubectl exec -it ${pod_name} -n ${namespace} -- sh
    ```

    4. Start using `tikv-ctl` in local mode. The default db path in the TiKV container is `/var/lib/tikv/db`:

    ```
    ./tikv-ctl --data-dir /var/lib/tikv size -r 2
    ```

11.5.2.3 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} ${pod_name} 10080:10080 &>/tmp/
    ↪ portforward-tidb.log &
```

Then you can use the `tidb-ctl`:

```
tidb-ctl schema in mysql
```

11.5.2.4 Use Helm

Helm is a package management tool for Kubernetes. The installation steps are as follows:

11.5.2.4.1 Install the Helm client

Refer to Helm official documentation to install the Helm client.

If the server does not have access to the Internet, you need to download Helm on a machine with Internet access, and then copy it to the server. Here is an example of installing the Helm client `3.4.1`:

```
wget https://get.helm.sh/helm-v3.4.1-linux-amd64.tar.gz
tar zxvf helm-v3.4.1-linux-amd64.tar.gz
```

After decompression, you can see the following files:

```
linux-amd64/
linux-amd64/README.md
linux-amd64/helm
linux-amd64/LICENSE
```

Copy the `linux-amd64/helm` file to the server and place it under the `/usr/local/bin/` directory.

Then execute `helm verison`. If the command outputs normally, the Helm installation is successful:

```
helm version
```

```
version.BuildInfo{Version:"v3.4.1", GitCommit:"
    ↪ c4e74854886b2efe3321e185578e6db9be0a6e29", GitTreeState:"clean",
    ↪ GoVersion:"go1.14.11"}
```

11.5.2.4.2 Configure the Helm repo

Kubernetes applications are packed as charts 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 back up 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 server or computer using the following command:

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

Then you can search the chart provided by PingCAP using the following command:

```
helm search repo pingcap
```

```
NAME                  CHART VERSION APP VERSION   DESCRIPTION
pingcap/tidb-backup  v1.1.15                       A Helm chart for TiDB
    ↪ Backup or Restore
pingcap/tidb-cluster v1.1.15                       A Helm chart for TiDB
    ↪ Cluster
pingcap/tidb-drainer v1.1.15                       A Helm chart for TiDB
    ↪ Binlog drainer.
pingcap/tidb-lightning v1.1.15                     A Helm chart for TiDB
    ↪ Lightning
pingcap/tidb-operator v1.1.15       v1.1.15        tidb-operator Helm chart
    ↪ for Kubernetes
pingcap/tikv-importer v1.1.15                      A Helm chart for TiKV
    ↪ Importer
```

When a new version of chart has been released, you can use `helm repo update` to update the repository cached locally:

```
helm repo update
```

11.5.2.4.3 Helm common operations

Common Helm operations include `helm install`, `helm upgrade`, and `helm uninstall`. 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.

Before performing the deploy, upgrade and deploy, you can view the deployed applications via `helm ls`:

```
helm ls
```

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 ${release_name} ${chart_name} --namespace=${namespace} --
    ↪ version=${chart_version} -f ${values_file}
```

- Upgrade (the upgrade can be either modifying the `chart-version` to upgrade to the latest chart version, or editing the `values.yaml` file to update the configuration):

```
helm upgrade ${release_name} ${chart_name} --version=${chart_version} -
    ↪ f ${values_file}
```

- Delete:

  To delete the application deployed by Helm, run the following command:

```
helm uninstall ${release_name} -n ${namespace}
```

For more information on Helm, refer to Helm Documentation.

11.5.2.4.4 Use Helm chart offline

If the server has no Internet access, you cannot configure the Helm repo to install the TiDB Operator component and other applications. At this time, you need to download the chart file needed for cluster installation on a machine with Internet access, and then copy it to the server.

Use the following command to download the chart file required for cluster installation:

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

Copy these chart files to the server and decompress them. You can use these charts to install the corresponding components by running the `helm install` command. Take `tidb-operator` as an example:

```
tar zxvf tidb-operator.v1.1.15.tgz
helm install ${release_name} ./tidb-operator --namespace=${namespace}
```

11.5.2.5 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.

11.6 Configure

11.6.1 TiDB Binlog Drainer Configurations in Kubernetes

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

11.6.1.1 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
`timezone`	Timezone configuration	`UTC`
`drainerName`	The name of Statefulset	`""`
`clusterName`	The name of the source TiDB cluster	`demo`
`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`

Parameter	Description	Default Value
`logLevel` ↪	The log level of the drainer process	`info`

Parameter	Description	Default Value
`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`

Parameter	Description	Default Value
`initialCommitTs`	Used to initialize a checkpoint if the drainer does not have one. The value is a string type, such as `"424364429251444742"`	`"-1"`
`tlsCluster.enabled`	Whether or not to enable TLS between clusters	`false`

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"
```

11.6.2 Configuration of tidb-cluster Chart

This document describes the configuration of the tidb-cluster chart.

Note:

> For TiDB Operator v1.1 and later versions, it is recommended not to use the tidb-cluster chart to deploy and manage the TiDB cluster. For details, refer to TiDB Operator v1.1 Notes.

11.6.2.1 Parameters

Parameter	Description	Default value
`rbac.create`	Whether to enable RBAC for Kubernetes	`true`

Parameter	Description	Default value
`clusterName`	TiDB cluster name. This variable is not set by default, and `tidb-cluster` directly uses `ReleaseName` during installation instead of the TiDB cluster name.	`nil`
`extraLabels`	Adds extra labels to the `TidbCluster` object (CRD). Refer to labels	`{}`

Parameter	Description	Default value
`schedulerName`	Name used by TiDB cluster	`tidb-scheduler`
`timezone`	The default time zone of the TiDB cluster	`UTC`
`pvReclaimPolicy`	Reclaim policy for PVs (Persistent Volumes) used by the TiDB cluster	`Retain`
`services[0].name`	Name of the service exposed by the TiDB cluster	`nil`

Parameter	Description	Default value
`services[0].type`	Type of the service exposed by the TiDB cluster (selected from `ClusterIP`, `NodePort`, `LoadBalancer`)	`nil`

Parameter	Description	Default value
`discovery.image`	Mirror image of the PD service discovery component in the TiDB cluster, which is used to provide service discovery for each PD instance to coordinate the startup sequence when the PD cluster is first started	`pingcap/tidb-operator:v1.0.0-beta.3`

Parameter	Description	Default value
`discovery.image`	Pull strategy for the PD service discovery component image	`IfNotPresent`
`discovery.resources.limits.cpu`	CPU resource limit for PD service discovery component	`250m`

Parameter	Description	Default value
`discovery.resources.limits.memory`	Memory resource limit for PD service discovery component	`150Mi`
`discovery.resources.requests.cpu`	CPU resource request for PD service discovery component	`80m`

Parameter	Description	Default value
`discovery.resources.memory`	Memory resource for PD service discovery component	`50Mi`
`discovery.requests`	request	

Parameter	Description	Default value
enableConfigMapRollout	When to enable automatic rolling update for the TiDB cluster. If enabled, the TiDB cluster automatically updates the corresponding components when the ConfigMap of the TiDB cluster changes. This configuration is only supported	false

Parameter	Description	Default value
`pd.config`	Configuration of PD in configuration file format. To view the default PD configuration file, refer to pd/conf/config.toml and select the tag of the corresponding PD version. To view the descriptions of parameters, refer	If the TiDB Operator version <= v1.0.0-beta.3, the default value is `nil` If the TiDB Operator version > v1.0.0-beta.3, the default value is `[log] level = "info" [replication] location-labels = ["region", "zone", "rack", "host"]`

Parameter	Description	Default value
pd.replicas	Number of Pods in PD	3
pd.image	PD image	pingcap.com/pd:v3.0.0-rc.1
pd.imagePullPolicy	Pull strategy for PD mirror	IfNotPresent
pd.logLevel	If TiDB Operator version> v1.0.0-beta.3, configure in pd.config: [log] level = "info"	info

Parameter	Description	Default value
`pd.storageClassName`	`storageClassName` is used by PD. `storageClassName` refers to a type of storage provided by the Kubernetes cluster. Different classes may be mapped to service quality levels, backup policies, or any policies determined by the cluster administrator.	`local-storage`

Parameter	Description	Default value
`pd.maxStoreDownTime`	`pd.maxStoreDownTime` refers to the time that a store node is disconnected before the node is marked as `down`. When the status becomes `down`, the store node starts to migrate its data to other store nodes.If TiDB Operator version > v1.0.0-beta.3, configure in	`30m`

Parameter	Description	Default value
`pd.maxReplicas`	`pd.maxReplicas` is the number of replicas in the TiDB cluster.If TiDB Operator version > v1.0.0-beta.3, configure in `pd.config:[replication] max-replicas =`	`3`
`pd.resources.limits.cpu`	CPU resource limit for each PD Pod	`nil`

Parameter	Description	Default value
`pd.resources.limits.memory`	Memory resources limit for each PD Pod	`nil`
`pd.resources.limits.storage`	Storage capacity limit for each PD Pod	`nil`
`pd.resources.requests.cpu`	CPU resources request for each PD Pod	`nil`
`pd.resources.requests.memory`	Memory resources request for each PD Pod	`nil`

Parameter	Description	Default value
`pd.resources.requests.storage`	Storage capacity requested for each PD Pod	`1Gi`
`pd.affinity`	`pd.affinity` defines PD scheduling rules and preferences. For details, refer to affinity-and-anti-affinity	`{}`

Parameter	Description	Default value
`pd.nodeSelector`	`pd.nodeSelector` makes sure that PD Pods are dispatched only to nodes that have this key-value pair as a label. For details, refer to nodes-elector	`{}`

Parameter	Description	Default value
`pd.tolerations`	`pd.tolerations` applies to PD Pods, allowing PD Pods to be dispatched to nodes with specified taints. For details, refer to taint-and-toleration	`{}`
`pd.annotations`	Add specific `annotations` to PD Pods	`{}`

Parameter	Description	Default value
`tikv.config`	Configuration of TiKV in configuration file format. To view the default TiKV configuration file, refer to tikv/etc/config-template.toml and select the tag of the corresponding TiKV version. To view the descriptions	If the TiDB Operator version <= v1.0.0-beta.3, the default value is `nil`. If the TiDB Operator version > v1.0.0-beta.3, the default value is `log-level = "info"`. For example: `config: | log-level = "info"`

Parameter	Description	Default value
`tikv.replicas`	Number of Pods in TiKV	`3`
`tikv.image`	Image of TiKV	`pingcap/tikv:v3.0.0-rc.1`
`tikv.imagePullPolicy`	Pull strategy for TiKV image	`IfNotPresent`
`tikv.logLevel`	TiKV log level. If the TiDB Operator version > v1.0.0-beta.3, configure in `tikv.config:log-level`	`info`

Parameter	Description	Default value
`tikv.storage.storageClassName`	used by TiKV. `storageClassName` refers to a type of storage provided by the Kubernetes cluster. Different classes may be mapped to service quality levels, backup policies, or any policies determined by the cluster administrator.	`local-storage`

Parameter	Description	Default value
`tikv.syncLog`	indicates whether to enable the raft log replication feature. If enabled, the feature ensure that data is not lost during poweroff.If the TiDB Operator version > v1.0.0-beta.3, configure in `tikv.config:[raftstore]sync log =`	`true`

Parameter	Description	Default value
`tikv.grpcConcurrency`	Configure the concurrency size of the gRPC server thread pool. If the TiDB Operator version > v1.0.0-beta.3, configure in `tikv.config.server.grpc-concurrency` = 4	4
`tikv.resources.limits.cpu`	CPU resource limit for each TiKV Pod	`nil`

Parameter	Description	Default value
`tikv.resources.limits.memory`	Memory resource limit for each TiKV Pod	`nil`
`tikv.resources.limits.storage`	Storage capacity limit for each TiKV Pod	`nil`
`tikv.resources.limits.cpu`	CPU resource limit for each TiKV Pod	`nil`
`tikv.resources.requests.cpu`	CPU resource request for each TiKV Pod	`nil`
`tikv.resources.requests.memory`	Memory resource request for each TiKV Pod	`nil`

Parameter	Description	Default value
`tikv.resources.requests.storage`	Storage capacity requested for each TiKV Pod	`10Gi`
`tikv.affinity`	`tikv.affinity` defines TiKV's scheduling rules and preferences. For details, refer to affinity-and-anti-affinity	`{}`

Parameter	Description	Default value
`tikv.nodeSelector`	`tikv.nodeSelector` makes sure that TiKV Pods are dispatched only to nodes that have this key-value pair as a label. For details, refer to nodes-elector	`{}`

Parameter	Description	Default value
`tikv.tolerations`	`tikv.tolerations` applies to TiKV Pods, allowing TiKV Pods to be dispatched to nodes with specified taints. For details, refer to taint-and-toleration	`{}`
`tikv.annotations`	Add specific `annotations` for TiKV Pods	`{}`

Parameter	Description	Default value
`tikv.defaultcf.block-cache-size`	Specifies the block cache size. The block cache is used to cache uncompressed blocks. A large block cache setting can speed up reading. Generally it is recommended to set the block cache size to 30%-50% of `tikv.resources.limits.memory`. If	`1GB`

Parameter	Description	Default value
`tikv.writecf.block-cache-size`	Specifies the block cache size of `writecf`. It is generally recommended to be 10%-30% of `tikv.resources.limits.memory`. If the TiDB Operator version > v1.0.0-beta.3, configure in `tikv.config.rocksdb.writecf.block-cache-size =`	`256MB`

Parameter	Description	Default value
`tikv.readpool.storage.high-concurrency`	Size of TiKV storage thread pool for high-/medium/low priority operations	`4`
`tikv.readpool.storage.normal-concurrency`	If the TiDB Operator version > v1.0.0-beta.3, configure in `tikv.config.readpool.storage.high-concurrency = 4`	`4`
`tikv.readpool.storage.low-concurrency`	`tikv.config.readpool.storage.normal-concurrency = 4`	`4`

Parameter	Description	Default value
`tikv.readpoolCoprocessorConcurrency`	Usually, `tikv.resources.limits.cpu` > 8, set `tikv.readpoolCoprocessorConcurrency` to `tikv.resources.limits.cpu` * 0.8If the TiDB Operator version > v1.0.0-beta.3, configure in `tikv.config:[readpool.coprocessor]high-concurrency = 8normal-concurrency`	

Parameter	Description	Default value
`tikv.storage.scheduler-worker-pool-size`	Size of the scheduler working pool of the TiKV scheduler, which should be increased in the case of rewriting, and should be smaller than the total CPU cores.If the TiDB Operator version > v1.0.0-beta.3, configure in `tikv.config.storage.scheduler-worker-`	4

Parameter	Description	Default value
`tidb.config`	Configuration of TiDB in configuration file format. To view the default TiDB configuration file, refer to configuration file and select the tag of the corresponding TiDB version. To view the descriptions of parameters, refer to	If the TiDB Operator version <= v1.0.0-beta.3, the default value is:`nil`If the TiDB Operator version > v1.0.0-beta.3, the default value is:`[log] level = "info" `For example: `config: | [log] level = "info" `

Parameter	Description	Default value
`tidb.replicas`	Number of Pods in TiDB	`2`
`tidb.image`	Image of TiDB	`pingcap/tidb:v3.0.0-rc.1`
`tidb.imagePullPolicy`	Pull strategy for TiDB image	`IfNotPresent`
`tidb.logLevel`	TiDB log level. If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config.toml`	`info`

Parameter	Description	Default value
`tidb.resources.limits.cpu`	CPU resource limit for each TiDB Pod	`nil`
`tidb.resources.limits.memory`	Memory resource limit for each TiDB Pod	`nil`
`tidb.resources.requests.storage`	Storage capacity limit for each TiDB Pod	`nil`
`tidb.resources.requests.cpu`	CPU resource request for each TiDB Pod	`nil`
`tidb.resources.requests.memory`	Memory resource request for each TiDB Pod	`nil`

Parameter	Description	Default value
`tidb.passwordSecretName`	The name of the Secret which stores the TiDB username and password. This Secret can be created with the following command: `kubectl create secret generic tidb secret --from-literal=root=${password} --namespace=${namespace}`	`nil`

Parameter	Description	Default value
`tidb.initSql`	The initialization script that will be executed after the TiDB cluster is successfully started.	`nil`
`tidb.affinity`	defines the scheduling rules and preferences of TiDB. For details, refer to affinity-and-anti-affinity.	`{}`

Parameter	Description	Default value
`tidb.nodeSelector`	`tidb.nodeSelector` makes sure that TiDB Pods are only dispatched to nodes that have this key-value pair as a label. For details, refer to nodeselector.	`{}`

Parameter	Description	Default value
`tidb.tolerations`	`tidb.tolerations` applies to TiKV Pods, allowing TiKV Pods to be dispatched to nodes with specified taints. For details, refer to taint-and-toleration.	`{}`
`tidb.annotations`	Add specific `annotations` for TiDB Pods	`{}`

Parameter	Description	Default value
`tidb.maxFailoverCount`	The maximum number of failover in TiDB. If it is set to 3, then TiDB supports at most 3 instances failover at the same time.	`3`
`tidb.service.type`	Type of the service exposed by TiDB	`NodePort`

Parameter	Description	Default value
`tidb.service.externalTrafficPolicy`	Indicates whether this service wants to route external traffic to a local node or a cluster-wide endpoint. Two options are available: `Cluster` (default) and `Local`. `Cluster` hides the client's source IP. In such case, the traffic might need to be redi-	`nil`

Parameter	Description	Default value
`tidb.service.loadBalancerIP`	Specifies TiDB load balancing IP. Some cloud service providers allow you to specify the loadBalancer IP. In these cases, a load balancer is created using a user-specified loadBalancerIP. If the loadBalancerIP field is not specified, the loadBalancer is set with a	`nil`

Parameter	Description	Default value
`tidb.service.mysqlNodePort`	MySQL Node-Port exposed by the TiDB service	/
`tidb.service.exposeStatus`	Whether the TiDB status service exposes the status port	`true`
`tidb.service.statusNodePort`	Specifies the NodePort where the status port of the TiDB service is exposed	/

Parameter	Description	Default value
`tidb.separateSlowLog`	Whether to output TiDB's SlowLog in a standalone container in sidecar mode	If the TiDB Operator version <= v1.0.0-beta.3, the default value is `false`. If the TiDB Operator version > v1.0.0-beta.3, the default value is `true`.

Parameter	Description	Default value
`tidb.slowLogTailer.image`	image of TiDB. `slowLogTailer` is a sidecar type container used to output TiDB's SlowLog. This configuration only takes effect when `tidb.separateSlowLog = true`.	busybox:1.26.2
`tidb.slowLogTailer.resources.limits.cpu`	CPU source limit for slowLogTailer of each TiDB Pod	100m

Parameter	Description	Default value
`tidb.slowLogTailer.resources.limits.memory`	Memory limit for slowLogTailer of each TiDB Pod	`50Mi`
`tidb.slowLogTailer.resources.requests.cpu`	CPU request for slowLogTailer of each TiDB Pod	`20m`
`tidb.slowLogTailer.resources.requests.memory`	Memory request for slowLogTailer of each TiDB Pod	`5Mi`

Parameter	Description	Default value
`tidb.plugin.enable`	Whether to enable the TiDB plugin feature	`false`
`tidb.plugin.directory`	Specifies the directory of the TiDB plugin	`/plugins`
`tidb.plugin.list`	Specifies the list of plugins loaded by TiDB. Rules of naming Plugin ID: [plugin name]-version. For example: 'conn_limit-1'	`[]`

Parameter	Description	Default value
`tidb.prepared-plan-cache.enabled`	Whether to enable the prepared plan cache of TiDB. If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:[prepared-plan-cache]enabled =`	`false`

Parameter	Description	Default value
`tidb.prepared-plan-cache.capacity`	Number of prepared plan cache of TiDB.If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:[prepared-plan-cache]capacity =`	`100`

Parameter	Description	Default value
`tidb.txnLocalLatchesEnabled`	Whether to enable the transaction memory lock. It is recommended to enable the transaction memory lock when there are many local transaction conflicts.If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:[txn.local-`	`false`

Parameter	Description	Default value
`tidb.txnLocalLatchesCapacity`	Capacity of the transaction memory lock. The number of slots corresponding to the hash is automatically adjusted up to an exponential multiple of 2. Every slot occupies 32 bytes of memory. When the range of data to be written is wide	`10240000`

Parameter	Description	Default value
`tidb.token-limit`	Limit on concurrent sessions executed by TiDB If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:token-limit`	`1000`

Parameter	Description	Default value
`tidb.memQuotaQuery`	Memory limit for TiDB query. 32GB by default.If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:mem-quota-query` =	`34359738368`

Parameter	Description	Default value
`tidb.checkMb4ValueInUtf8`	Determines whether to check `mb4` characters when the current character set is `utf8`. If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:check-mb4-value-in-utf8 = true`	`true`

Parameter	Description	Default value
`tidb.treatOldVersionUtf8AsUtf8mb4`	Used for upgrade compatibility. If this parameter is set to `true`, the `utf8` character set in the old version of tables/columns is treated as the `utf8mb4` character set.If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:treat-`	`true`

Parameter	Description	Default value
`tidb.lease`	Term of TiDB Schema lease. It is very dangerous to change this parameter, so do not change it unless you know the possible consequences.If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:lease = "45s"`	`45s`

Parameter	Description	Default value
`tidb.maxProcs`	The maximum number of CPU cores that are available. 0 represents the total number of CPUs on the machine or Pod.If the TiDB Operator version > v1.0.0-beta.3, configure in `tidb.config:[performance] max-procs =`	`0`

Parameter	Description	Default value

11.6.3 Configuration of tidb-backup Chart

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

> **Note:**
>
> For TiDB Operator v1.1 and later versions, it is recommended not to use the tidb-backup chart to deploy and manage the TiDB cluster backup. For details, refer to Notes for TiDB Operator v1.1.

11.6.3.1 Configuration

11.6.3.1.1 `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)

11.6.3.1.2 `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

11.6.3.1.3 `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.

11.6.3.1.4 `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}
```

11.6.3.1.5 `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.

11.6.3.1.6 `storage.size`

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

11.6.3.1.7 `backupOptions`

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

11.6.3.1.8 `restoreOptions`

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

11.6.3.1.9 `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.

11.6.3.1.10 `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
```

11.6.3.1.11 `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.

11.6.3.1.12 `ceph.bucket`

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

11.6.3.1.13 `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}
```

11.7 TiDB Log Collection in Kubernetes

The system and application logs can be useful for troubleshooting issues and automating operations. This article briefly introduces the methods of collecting logs of TiDB and its related components.

11.7.1 Collect logs of TiDB and Kubernetes components

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 in which Kubernetes is located in the collection scope by common methods; 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.

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 charged log collection options that you can choose from.

11.7.2 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 charged log collection options that you can choose from.

11.8 Monitoring and Alerts on Kubernetes

This document describes how to monitor a Kubernetes cluster and configure alerts for the cluster.

11.8.1 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, or TiDB Operator. To monitor these components or resources, you need to deploy a monitoring system across the entire Kubernetes cluster.

11.8.1.1 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 if 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 have 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.

11.8.1.2 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. 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.

11.8.1.3 Alerts in Kubernetes

If you deploy a monitoring system for Kubernetes hosts and services using 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.

12 Release Notes

12.1 v1.1

12.1.1 TiDB Operator 1.1.15 Release Notes

Release date: February 17, 2022

TiDB Operator version: 1.1.15

12.1.1.1 Bug Fixes

- Fix a potential goroutine leak when TiDB Operator checks the Region leader count of TiKV (#4291, [@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.2 TiDB Operator 1.1.14 Release Notes

Release date: October 21, 2021

TiDB Operator version: 1.1.14

12.1.2.1 Bug Fixes

- Fix the security vulnerabilities in the `tidb-backup-manager` and `tidb-operator` images (#4217, [@KanShiori](https://github.com/KanShiori))

12.1.3 TiDB Operator 1.1.13 Release Notes

Release date: July 2, 2021

TiDB Operator version: 1.1.13

12.1.3.1 Improvements

- Support configuring TLS certificates for TiCDC sinks (#3926, [@handlerww](https://github.com/han
- Support using the TiKV version as the tag for BR `toolImage` if no tag is specified (#4048, [@KanShiori](https://github.com/KanShiori))
- Support handling PVC during scaling of TiDB (#4033, [@csuzhangxc](https://github.com/csuzhangx
- Support masking the backup password in logging (#3979, [@dveeden](https://github.com/dveeden))

12.1.3.2 Bug Fixes

- Fix the issue that TiDB Operator may panic during the deployment of heterogeneous clusters (#4054 #3965, [@KanShiori](https://github.com/KanShiori) [@july2993](https://github.com/july2993))
- Fix the issue that TiDB instances are kept in TiDB Dashboard after being scaled in (#3929, [@july2993](https://github.com/july2993))

12.1.4 TiDB Operator 1.1.12 Release Notes

Release date: April 15, 2021

TiDB Operator version: 1.1.12

12.1.4.1 New Features

- Support setting customized environment variables for backup and restore job containers (#3833, [@dragonly](https://github.com/dragonly))
- Add additional volume and volumeMount configurations to TidbMonitor (#3855, [@mikechengwei](https://github.com/mikechengwei))
- The resources in the tidb-operator chart use the new service account when `appendReleaseSuffix` is set to `true` (#3819, [@DanielZhangQD](https://github.com/DanielZhangQ

12.1.4.2 Improvements

- Add retry for DNS lookup failure exception in TiDBInitializer (#3884, [@handlerww](https://github.com/handlerww))
- Support multiple PVCs for PD during scaling and failover (#3820, [@dragonly](https://github.com/dragonly))
- Optimize the `PodsAreChanged` function (#3901, [@shonge](https://github.com/shonge))

12.1.4.3 Bug Fixes

- Fix the issue that PVCs will be set to incorrect size if multiple PVCs are configured for PD/TiKV (#3858, [@dragonly](https://github.com/dragonly))
- Fix the panic issue when `.spec.tidb` is not set in the TidbCluster CR with TLS enabled (#3852, [@dragonly](https://github.com/dragonly))
- Fix the wrong PVC status in `UnjoinedMembers` for PD and DM (#3836, [@dragonly](https://github.com/dragonly))

12.1.5 TiDB Operator 1.1.11 Release Notes

Release date: February 26, 2021

TiDB Operator version: 1.1.11

12.1.5.1 New Features

- Support configuring durations for leader election (#3794, [@july2993](https://github.com/july2993))
- Support setting customized store labels according to the node labels (#3784, [@L3T](https://github.com/L3T))

12.1.5.2 Improvements

- Add TiFlash rolling upgrade logic to avoid all TiFlash stores being unavailable at the same time during the upgrade (#3789, [@handlerww](https://github.com/handlerww))
- Retrieve the region leader count from TiKV Pod directly instead of from PD to get the accurate count (#3801, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Print RocksDB and Raft logs to stdout to support collecting and querying the logs in Grafana (#3768, [@baurine](https://github.com/baurine))

12.1.6 TiDB Operator 1.1.10 Release Notes

Release date: January 28, 2021

TiDB Operator version: 1.1.10

12.1.6.1 Compatibility Changes

- Due to the changes of #3638, the `apiVersion` of ClusterRoleBinding, ClusterRole, RoleBinding, and Role created in the TiDB Operator chart is changed from `rbac ↪ .authorization .k8s.io/v1beta1` to `rbac.authorization.k8s.io/v1`. In this case, upgrading TiDB Operator through `helm upgrade` may report the following error:

```
Error: UPGRADE FAILED: rendered manifests contain a new resource that
↪  already exists. Unable to continue with update: existing resource
↪  conflict: namespace:, name: tidb-operator:tidb-controller-manager
↪ , existing_kind: rbac.authorization.k8s.io/ v1, Kind=ClusterRole,
↪ new_kind: rbac.authorization.k8s.io/v1, Kind=ClusterRole
```

For details, refer to helm/helm#7697. In this case, you need to delete TiDB Operator through `helm uninstall` and then reinstall it (deleting TiDB Operator will not affect the current TiDB clusters).

12.1.6.2 Rolling Update Changes

- Upgrading TiDB Operator will cause the recreation of the TidbMonitor Pod due to #3684

12.1.6.3 New Features

- Support canary upgrade of TiDB Operator (#3548, [@shonge](https://github.com/shonge), #3554, [@cvvz](https://github.com/cvvz))
- TidbMonitor supports `remotewrite` configuration (#3679, [@mikechengwei](https://github.com/mik
- Support configuring init containers for components in the TiDB cluster (#3713, [@handlerww](https://github.com/handlerww))
- Add local backend support to the TiDB Lightning chart (#3644, [@csuzhangxc](https://github.com/c

12.1.6.4 Improvements

- Support customizing the storage config for TiDB slow log (#3731, [@BinChenn](https://github.com/B
- Add the `tidb_cluster` label for the scrape jobs in TidbMonitor to support monitoring multiple clusters (#3750, [@mikechengwei](https://github.com/mikechengwei))
- Supports persisting checkpoint for the TiDB Lightning helm chart (#3653, [@csuzhangxc](https://github.com/csuzhangxc))
- Change the directory of the customized alert rules in TidbMonitor from `tidb:${ ↪ tidb_image_version}` to `tidb:${initializer_image_version}` so that when the TiDB cluster is upgraded afterwards, the TidbMonitor Pod will not be recreated (#3684, [@BinChenn](https://github.com/BinChenn))

12.1.6.5 Bug Fixes

- Fix the issue that when TLS is enabled for the TiDB cluster, if `spec.from` or `spec.to` is not configured, backup and restore jobs with BR might fail (#3707, [@BinChenn](https://github.com/BinChenn))

- Fix the bug that if the advanced StatefulSet is enabled and `delete-slots` annotations are added for PD or TiKV, the Pods whose ordinal is bigger than `replicas - 1` will be terminated directly without any pre-delete operations such as evicting leaders (#3702, [@cvvz](https://github.com/cvvz))
- Fix the issue that after the Pod has been evicted or killed, the status of backup or restore is not updated to `Failed` (#3696, [@csuzhangxc](https://github.com/csuzhangxc))
- Fix the issue that when the TiKV cluster is not bootstrapped due to incorrect configuration, the TiKV component could not be recovered by editing `TidbCluster` CR (#3694, [@cvvz](https://github.com/cvvz))

12.1.7 TiDB Operator 1.1.9 Release Notes

Release date: December 28, 2020

TiDB Operator version: 1.1.9

12.1.7.1 Improvements

- Support `spec.toolImage` for `Backup` & `Restore` to define the image used to provide the Dumpling/TiDB Lightning binary executables (#3641, [@BinChenn](https://github.com/BinChenn)

12.1.7.2 Bug Fixes

- Fix the issue that Prometheus can't pull metrics for TiKV Importer (#3631, [@csuzhangxc](https://github.com/csuzhangxc))
- Fix the compatibility issue for using BR to back up/restore from/to GCS (#3654, [@dragonly](https://github.com/dragonly))

12.1.8 TiDB Operator 1.1.8 Release Notes

Release date: December 21, 2020

TiDB Operator version: 1.1.8

12.1.8.1 New Features

- Support arbitrary Volume and VolumeMount for `PD`, `TiDB`, `TiKV`, `TiFlash`, `Backup` and `Restore`, which enables using NFS or any other kubernetes supported volume source for backup/restore workflow (#3517, [@dragonly](https://github.com/dragonly))

12.1.8.2 Improvements

- Support cluster and client TLS for `tidb-lightning` and `tikv-importer` helm charts (#3598, [@csuzhangxc](https://github.com/csuzhangxc))
- Support setting additional ports for the TiDB service. Users can utilize this feature to implement customized services, e.g. additional health check (#3599, [@handlerww](https://github.com/handlerww))
- Support skipping TLS when connecting `TidbInitializer` to TiDB Server (#3564, [@LinuxGit](https://github.com/LinuxGit))
- Support tableFilters for restoring data using TiDB Lightning (#3521, [@sstubbs](https://github.com/
- Support Prometheus to scrape metrics data from multiple TiDB clusters (#3622, [@mikechengwei](https://github.com/mikechengwei))

  ACTION REQUIRED: If `TidbMonitor` CRs are deployed, update the `spec.initializer.version` to `v4.0.9` after upgrading TiDB Operator to v1.1.8, or some metrics will not be shown correctly in the Grafana dashboards. Prometheus crape job names are changed from `${component}` to `${namespace}-${TidbClusterName}-${component}`.
- `component` label is added to the scrape jobs of Prometheus in `TidbMonitor` (#3609, [@mikechengwei](https://github.com/mikechengwei))

12.1.8.3 Bug Fixes

- Fix the issue that TiDB cluster fails to deploy if `spec.tikv.storageVolumes` is configured (#3586, [@mikechengwei](https://github.com/mikechengwei))
- Fix codecs error for non-ASCII character password in the `TidbInitializer` job (#3569, [@handlerww](https://github.com/handlerww))
- Fix the issue that TiFlash Pods are misrecognized as TiKV Pods. The original issue can potentially cause TiDB Operator to scale in TiKV Pods to a number smaller than `tikv.replicas`, when there are TiFlash Pods in the `TidbCluster` (#3514, [@handlerww](https://github.com/handlerww))
- Fix the issue that deploying `Backup` CR without `spec.from` configured will crash `tidb-controller-manager` Pod when TLS is enabled for TiDB client (#3535, [@dragonly](https://github.com/dragonly))
- Fix the issue that TiDB Lightning doesn't log to stdout (#3617, [@csuzhangxc](https://github.com/cs

12.1.9 TiDB Operator 1.1.7 Release Notes

Release date: November 13, 2020

TiDB Operator version: 1.1.7

12.1.9.1 Compatibility Changes

- The behavior of `prometheus.spec.config.commandOptions` has changed. Any duplicated flags must be removed, or Prometheus will fail to start. (#3390, [@mightyguava](https://github.com/mightyguava))

  Flags that CANNOT be set are:

  - `--web.enable-admin-api`
  - `--web.enable-lifecycle`
  - `--config.file`
  - `--storage.tsdb.path`
  - `--storage.tsdb.retention`

12.1.9.2 New Features

- Support `spec.toolImage` for the `Backup` and `Restore` CR with BR to define the image used to provide the BR binary executables. Defaults to `pingcap/br:${tikv_version}` (#3471, [@namco1992](https://github.com/namco1992))
- Add `spec.tidb.storageVolumes`, `spec.tikv.storageVolumes`, and `spec.pd.storageVolumes` to support mounting multiple PVs for TiDB, TiKV, and PD (#3425 #3444, [@mikechengwei](https://github.com/mikechengwei))
- Add `spec.tidb.readinessProbe` config to support requesting `http://127.0.0.0:10080/status` for TiDB's readiness probe, TiDB version >= v4.0.9 required (#3438, [@july2993](https://github.com/july2993))
- Support PD leader transfer with advanced StatefulSet controller enabled (#3395, [@tangwz](https://github.com/tangwz))
- Support setting `OnDelete` update strategies for the StatefulSets via `spec.statefulSetUpdateStrategy` in the TidbCluster CR (#3408, [@cvvz](https://github.com/cvvz))
- Support HA scheduling when failover happens (#3419, [@cvvz](https://github.com/cvvz))
- Support smooth migration from TiDB clusters deployed using TiDB Ansible or TiUP or deployed in the same Kubernetes cluster to a new TiDB cluster (#3226, [@cvvz](https://github.com/cvvz))
- tidb-scheduler supports advanced StatefulSet (#3388, [@cvvz](https://github.com/cvvz))

12.1.9.3 Improvements

- Forbid to scale in TiKV when the number of UP stores is equal to or less than 3 (#3367, [@cvvz](https://github.com/cvvz))
- `phase` is added in `BackupStatus` and `RestoreStatus`, which will be in sync with the latest condition type and shown when doing `kubectl get` (#3397, [@namco1992](https://github.com/namco1992))
- Skip setting `tikv_gc_life_time` via SQL for backup and restore with BR when the TiKV version >= v4.0.8 (#3443, [@namco1992](https://github.com/namco1992))

12.1.9.4 Bug Fixes

- Fix the issue that PD cannot scale in to zero if there are other PD members outside of this `TidbCluster` (#3456, [@dragonly](https://github.com/dragonly))

12.1.10 TiDB Operator 1.1.6 Release Notes

Release date: October 16, 2020

TiDB Operator version: 1.1.6

12.1.10.1 Compatibility Changes

- With #3342, the `spec.pd.config` will be migrated from YAML format to TOML format automatically; however, if the following parameters are configured in the `spec.pd.config`, the migration cannot be done after upgrading TiDB Operator to v1.1.6. Therefore, please edit the `TidbCluster` CR to change the value of the parameter from string format to bool format, for example, from `"true"` to `true`.
  - `replication.strictly-match-label`
  - `replication.enable-placement-rules`
  - `schedule.disable-raft-learner`
  - `schedule.disable-remove-down-replica`
  - `schedule.disable-replace-offline-replica`
  - `schedule.disable-make-up-replica`
  - `schedule.disable-remove-extra-replica`
  - `schedule.disable-location-replacement`
  - `schedule.disable-namespace-relocation`
  - `schedule.enable-one-way-merge`
  - `schedule.enable-cross-table-merge`
  - `pd-server.use-region-storage`

12.1.10.2 Rolling Update Changes

- If `tidb.pingcap.com/sysctl-init: "true"` is set for `spec.tidb.annotations` or `spec.tikv.annotations`, the TiDB or TiKV cluster will be rolling updated after upgrading TiDB Operator to v1.1.6 due to #3305.
- If TiFlash is deployed, the TiFlash cluster will be rolling updated after upgrading TiDB Operator to v1.1.6 due to #3345.

12.1.10.3 New Features

- Add `spec.br.options` to the Backup and Restore CR to support customizing arguments for BR (#3360, [@lichunzhu](https://github.com/lichunzhu))
- Add `spec.tikv.evictLeaderTimeout` to TidbCluster CR to make TiKV evict leader timeout configurable (#3344, [@lichunzhu](https://github.com/lichunzhu))
- Support monitoring multiple TiDB clusters with one TidbMonitor CR when TLS is disabled. `spec.clusterScoped` is added to the TidbMonitor CR and needs to be set to `true` to monitor multiple clusters (#3308, [@mikechengwei](https://github.com/mikechengwei))
- Support specifying resources for all initcontainers (#3305, [@shonge](https://github.com/shonge))
- Support deploying heterogeneous TiDB clusters (#3003 #3009 #3113 #3155 #3253, [@mikechengwei](https://github.com/mikechengwei))

12.1.10.4 Improvements

- Support passing raw TOML config for TiFlash (#3355, [@july2993](https://github.com/july2993))
- Support passing raw TOML config for TiKV/PD (#3342, [@july2993](https://github.com/july2993))
- Support passing raw TOML config for TiDB (#3327, [@july2993](https://github.com/july2993))
- Support passing raw TOML config for Pump (#3312, [@july2993](https://github.com/july2993))
- Print proxy log of TiFlash to stdout (#3345, [@lichunzhu](https://github.com/lichunzhu))
- Add timestamp to the prefix of scheduled backup on GCS (#3340, [@lichunzhu](https://github.com/lichunzhu))
- Remove the apiserver and related packages (#3298, [@lonng](https://github.com/lonng))
- Remove the PodRestarter controller and `tidb.pingcap.com/pod-defer-deleting` annotation (#3296, [@lonng](https://github.com/lonng))
- Use BR metadata to get the total backup size (#3274, [@lichunzhu](https://github.com/lichunzhu))

12.1.10.5 Bug Fixes

- Fix the problem that may bootstrap multiple PD clusters (#3365, [@lichunzhu](https://github.com/lichunzhu))

12.1.11 TiDB Operator 1.1.5 Release Notes

Release date: September 18, 2020

TiDB Operator version: 1.1.5

12.1.11.1 Compatibility Changes

- If the TiFlash version is earlier than `v4.0.5`, please set `spec.tiflash.config.config.flash.service_addr: {clusterName}-tiflash-POD_NUM.{clusterName}-tiflash-peer.{namespace}.svc:3930` in the TidbCluster CR (`{clusterName}`

and `{namespace}` need to be replaced with the real value) before upgrading TiDB Operator to v1.1.5 or later versions. When TiFlash is going to be upgraded to `v4.0.5` or later versions, please remove `spec.tiflash.config.config.flash.service_addr` in the TidbCluster CR at the same time (#3191, [@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.11.2 New Features

- Support configuring `serviceAccount` for TiDB/Pump/PD (#3246, [@july2993](https://github.com/
- Support configuring `spec.tikv.config.log-format` and `spec.tikv.config.server.max-grpc-send-msg-len` (#3199, [@kolbe](https://github.com/kolbe))
- Support labels configuration for TiDB (#3188, [@howardlau1999](https://github.com/howardlau1999
- Support recovery from failover for TiFlash and TiKV (#3189, [@DanielZhangQD](https://github.com
- Add `mountClusterClientSecret` configuration for PD and TiKV. If the config is set to `true`, TiDB Operator will mount the `${cluster_name}-cluster-client-secret` to the PD or TiKV containers (#3282, [@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.11.3 Improvements

- Adapt TiDB/PD/TiKV configurations to v4.0.6 (#3180, [@lichunzhu](https://github.com/lichunzhu
- Support mounting the cluster client certificate to PD pod (#3248, [@weekface](https://github.com/weekface))
- Scaling takes precedence over upgrading for TiFlash/PD/TiDB. This is to avoid that Pods cannot be scaled in case of upgrade failure (#3187, [@lichunzhu](https://github.com/lichunzhu))
- Support the `imagePullSecrets` configuration for Pump (#3214, [@weekface](https://github.com/we
- Update the default configuration for TiFlash (#3191, [@DanielZhangQD](https://github.com/Daniel
- Remove `ClusterRole` from TidbMonitor CR (#3190, [@weekface](https://github.com/weekface))
- Drainer that are deployed by Helm and exits normally will no longer be restarted (#3151, [@lichunzhu](https://github.com/lichunzhu))
- The tidb-scheduler HA strategy takes failover pods into consideration (#3171, [@cofyc](https://github.com/cofyc))

12.1.11.4 Bug Fixes

- Fix the problem that the `Env` settings are ignored for the Grafana container in the TidbMonitor CR (#3237, [@tirsen](https://github.com/tirsen))

12.1.12 TiDB Operator 1.1.4 Release Notes

Release date: August 21, 2020

TiDB Operator version: 1.1.4

12.1.12.1 Notable changes

- `TableFilter` is added to the `BackupSpec` and `RestoreSpec`. `TableFilter` supports backing up specific databases or tables with Dumpling or BR and supports restoring specific databases or tables with BR. `BackupSpec.Dumpling.TableFilter` is deprecated since v1.1.4. Please configure `BackupSpec.TableFilter` instead. Since TiDB v4.0.3, you can configure `BackupSpec.TableFilter` to replace the `BackupSpec.BR.DB` and `BackupSpec.BR.Table` fields and configure `RestoreSpec.TableFilter` to replace the `RestoreSpec.BR.DB` and `RestoreSpec.BR.Table` fields (#3134, [@sstubbs](https://github.com/sstubbs))
- Update the version of TiDB and tools to v4.0.4 (#3135, [@lichunzhu](https://github.com/lichunzhu))
- Support customizing environment variables for the initializer container in the TidbMonitor CR (#3109, [@kolbe](https://github.com/kolbe))
- Support patching PVCs when the storage request is increased (#3096, [@cofyc](https://github.com/cofyc))
- Support TLS for Backup & Restore with Dumpling & TiDB Lightning (#3100, [@lichunzhu](https://github.com/lichunzhu))
- Support `cert-allowed-cn` for TiFlash (#3101, [@DanielZhangQD](https://github.com/DanielZhang
- Add support for the `max-index-length` TiDB config option to the TidbCluster CRD (#3076, [@kolbe](https://github.com/kolbe))
- Fix goroutine leak when TLS is enabled (#3081, [@DanielZhangQD](https://github.com/DanielZhan
- Fix a memory leak issue caused by etcd client when TLS is enabled (#3064, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Support TLS for TiFlash (#3049, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Configure TZ environment for admission webhook and advanced statefulset controller deployed in tidb-operator chart (#3034, [@cofyc](https://github.com/cofyc))

12.1.13 TiDB Operator 1.1.3 Release Notes

Release date: July 27, 2020

TiDB Operator version: 1.1.3

12.1.13.1 Action Required

- Add a field `cleanPolicy` in `BackupSpec` to denote the clean policy for backup data when the Backup CR is deleted from the cluster (default to `Retain`). Note that before v1.1.3, TiDB Operator will clean the backup data in the remote storage when the Backup CR is deleted, so if you want to clean backup data as before, set `spec.cleanPolicy` in `Backup` CR or `spec.backupTemplate.cleanPolicy` in `BackupSchedule` CR to `Delete`. (#3002, [@lichunzhu](https://github.com/lichunzhu))

- Replace `mydumper` with `dumpling` for backup.

If `spec.mydumper` is configured in the `Backup` CR or `spec.backupTemplate.mydumper` ↪ is configured in the `BackupSchedule` CR, migrate it to `spec.dumpling` or `spec.` ↪ `backupTemplate.dumpling`. After you upgrade TiDB Operator to v1.1.3, note that `spec.mydumper` or `spec.backupTemplate.mydumper` will be lost after the upgrade. (#2870, [@lichunzhu](https://github.com/lichunzhu))

12.1.13.2 Other Notable Changes

- Update tools in backup manager to v4.0.3 (#3019, [@lichunzhu](https://github.com/lichunzhu))
- Support `cleanPolicy` for the `Backup` CR to define the clean behavior of the backup data in the remote storage when the `Backup` CR is deleted (#3002, [@lichunzhu](https://github.com/lichunzhu))
- Add TLS support for TiCDC (#3011, [@weekface](https://github.com/weekface))
- Add TLS support between Drainer and the downstream database server (#2993, [@lichunzhu](https://github.com/lichunzhu))
- Support specifying `mysqlNodePort` and `statusNodePort` for TiDB Service Spec (#2941, [@lichunzhu](https://github.com/lichunzhu))
- Fix the `initialCommitTs` bug in Drainer's `values.yaml` (#2857, [@weekface](https://github.com/we
- Add `backup` config for TiKV server, add `enable-telemetry`, and deprecate `disable-` ↪ `telemetry` config for PD server (#2964, [@lichunzhu](https://github.com/lichunzhu))
- Add commitTS info column in `get restore` command (#2926, [@lichunzhu](https://github.com/lich
- Update the used Grafana version from v6.0.1 to v6.1.6 (#2923, [@lichunzhu](https://github.com/lichu
- Support showing commitTS in restore status (#2899, [@lichunzhu](https://github.com/lichunzhu))
- Exit without error if the backup data the user tries to clean does not exist (#2916, [@lichunzhu](https://github.com/lichunzhu))
- Support auto-scaling by storage for TiKV in `TidbClusterAutoScaler` (#2884, [@Yisaer](https://github.com/Yisaer))
- Clean temporary files in `Backup` job with `Dumpling` to save space (#2897, [@lichunzhu](https://github.com/lichunzhu))
- Fail the backup job if existing PVC's size is smaller than the storage request in the backup job (#2894, [@lichunzhu](https://github.com/lichunzhu))
- Support scaling and auto-failover even if a TiKV store fails in upgrading (#2886, [@cofyc](https://github.com/cofyc))
- Fix a bug that the `TidbMonitor` resource could not be set (#2878, [@weekface](https://github.com/weekface))
- Fix an error for the monitor creation in the tidb-cluster chart (#2869, [@8398a7](https://github.com/8
- Remove `readyToScaleThresholdSeconds` in `TidbClusterAutoScaler`; TiDB Operator won't support de-noise in `TidbClusterAutoScaler` (#2862, [@Yisaer](https://github.com/Yisaer
- Update the version of TiDB Lightning used in tidb-backup-manager from v3.0.15 to v4.0.2 (#2865, [@lichunzhu](https://github.com/lichunzhu))

12.1.14 TiDB Operator 1.1.2 Release Notes

Release date: July 1, 2020

TiDB Operator version: 1.1.2

12.1.14.1 Action Required

- An incompatible issue with PD 4.0.2 has been fixed. Please upgrade TiDB Operator to v1.1.2 before deploying TiDB 4.0.2 and later versions (#2809, [@cofyc](https://github.com/cofyc))

12.1.14.2 Other Notable Changes

- Collect metrics for TiCDC, TiDB Lightning and TiKV Importer (#2835, [@weekface](https://github.com/weekface))
- Update PD/TiDB/TiKV config to v4.0.2 (#2828, [@DanielZhangQD](https://github.com/DanielZha
- Fix the bug that `PD` Member might still exist after scaling-in (#2793, [@Yisaer](https://github.com/Y
- Support Auto-Scaler Reference in `TidbCluster` Status when there exists `TidbClusterAutoScaler` (#2791, [@Yisaer](https://github.com/Yisaer))
- Support configuring container lifecycle hooks and `terminationGracePeriodSeconds` in TiDB spec (#2810, [@weekface](https://github.com/weekface))

12.1.15 TiDB Operator 1.1.1 Release Notes

Release date: June 19, 2020

TiDB Operator version: 1.1.1

12.1.15.1 Notable changes

- Add the `additionalContainers` and `additionalVolumes` fields so that TiDB Operator can support adding sidecars to `TiDB`, `TiKV`, `PD`, etc. (#2229, [@yeya24](https://github.com/yeya24
- Add cross check to ensure TiKV is not scaled or upgraded at the same time (#2705, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Fix the bug that TidbMonitor will scrape multi TidbCluster with the same name in different namespaces when then namespace in `ClusterRef` is not set (#2746, [@Yisaer](https://github.com/Yisaer))
- Update TiDB Operator examples to deploy TiDB Cluster 4.0.0 images (#2600, [@kolbe](https://github.com/kolbe))
- Add the `alertMangerAlertVersion` option to TidbMonitor (#2744, [@weekface](https://github.com/weekface))
- Fix alert rules lost after rolling upgrade (#2715, [@weekface](https://github.com/weekface))
- Fix an issue that pods may be stuck in pending for a long time in scale-out after a scale-in (#2709, [@cofyc](https://github.com/cofyc))
- Add `EnableDashboardInternalProxy` in `PDSpec` to let user directly visit PD Dashboard (#2713, [@Yisaer](https://github.com/Yisaer))

- Fix the PV syncing error when `TidbMonitor` and `TidbCluster` have different values in `reclaimPolicy` (#2707, [@Yisaer](https://github.com/Yisaer))
- Update Configuration to v4.0.1 (#2702, [@Yisaer](https://github.com/Yisaer))
- Change tidb-discovery strategy type to `Recreate` to fix the bug that more than one discovery pod may exist (#2701, [@weekface](https://github.com/weekface))
- Expose the `Dashboard` service with `HTTP` endpoint whether `tlsCluster` is enabled (#2684, [@Yisaer](https://github.com/Yisaer))
- Add the `.tikv.dataSubDir` field to specify subdirectory within the data volume to store TiKV data (#2682, [@cofyc](https://github.com/cofyc))
- Add the `imagePullSecrets` attribute to all components (#2679, [@weekface](https://github.com/wee
- Enable StatefulSet and Pod validation webhook to work at the same time (#2664, [@Yisaer](https://github.com/Yisaer))
- Emit an event if it fails to sync labels to TiKV stores (#2587, [@PengJi](https://github.com/PengJi))
- Make `datasource` information hidden in log for `Backup` and `Restore` jobs (#2652, [@Yisaer](https://github.com/Yisaer))
- Support the `DynamicConfiguration` switch in TidbCluster Spec (#2539, [@Yisaer](https://github.co
- Support `LoadBalancerSourceRanges` in the `ServiceSpec` for the `TidbCluster` and `TidbMonitor` (#2610, [@shonge](https://github.com/shonge))
- Support `Dashboard` metrics ability for `TidbCluster` when `TidbMonitor` deployed (#2483, [@Yisaer](https://github.com/Yisaer))
- Bump the DM version to v2.0.0-beta.1 (#2615, [@tennix](https://github.com/tennix))
- support setting discovery resources (#2434, [@shonge](https://github.com/shonge))
- Support the Denoising for the `TidbCluster` Auto-scaling (#2307, [@vincent178](https://github.com/
- Support scraping `Pump` and `Drainer` metrics in TidbMonitor (#2750, [@Yisaer](https://github.com/Y

12.1.16 TiDB Operator 1.1 GA Release Notes

Release date: May 28, 2020

TiDB Operator version: 1.1.0

12.1.16.1 Upgrade from v1.0.x

For v1.0.x users, refer to Upgrade TiDB Operator to upgrade TiDB Operator in your cluster. Note that you should read the release notes (especially breaking changes and action required items) before the upgrade.

12.1.16.2 Breaking changes since v1.0.0

- Change TiDB pod `readiness` probe from `HTTPGet` to `TCPSocket` 4000 port. This will trigger rolling-upgrade for the `tidb-server` component. You can set `spec.paused` ↪ to `true` before upgrading tidb-operator to avoid the rolling upgrade, and set it back to `false` when you are ready to upgrade your TiDB server (#2139, [@weekface](https://github.com/weekface))

- `--advertise-address` is configured for `tidb-server`, which would trigger rolling-upgrade for the TiDB server. You can set `spec.paused` to `true` before upgrading TiDB Operator to avoid the rolling upgrade, and set it back to `false` when you are ready to upgrade your TiDB server (#2076, [@cofyc](https://github.com/cofyc))
- `--default-storage-class-name` and `--default-backup-storage-class-name` flags are abandoned, and the storage class defaults to Kubernetes default storage class right now. If you have set default storage class different than Kubernetes default storage class, set them explicitly in your TiDB cluster Helm or YAML files. (#1581, [@cofyc](https://github.com/cofyc))
- Add the `timezone` support for all charts (#1122, [@weekface](https://github.com/weekface)).

  For the `tidb-cluster` chart, we already have the `timezone` option (`UTC` by default). If the user does not change it to a different value (for example, `Asia/Shanghai`), none of the Pods will be recreated.

  If the user changes it to another value (for example, `Aisa/Shanghai`), all the related Pods (add a `TZ` env) will be recreated, namely rolling updated.

  The related Pods include `pump`, `drainer`, `discovery`, `monitor`, `scheduled backup`, `tidb-initializer`, and `tikv-importer`.

  All images' time zone maintained by TiDB Operator is `UTC`. If you use your own images, you need to make sure that the time zone inside your images is `UTC`.

12.1.16.3 Other Notable changes

- Fix `TidbCluster` upgrade bug when `PodWebhook` and `Advancend StatefulSet` are both enabled (#2507, [@Yisaer](https://github.com/Yisaer))
- Support preemption in `tidb-scheduler` (#2510, [@cofyc](https://github.com/cofyc))
- Update BR to v4.0.0-rc.2 to include the `auto_random` fix (#2508, [@DanielZhangQD](https://github.
- Supports advanced statefulset for TiFlash (#2469, [@DanielZhangQD](https://github.com/DanielZh
- Sync Pump before TiDB (#2515, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Improve performance by removing `TidbControl` lock (#2489, [@weekface](https://github.com/weekf
- Support TiCDC in `TidbCluster` (#2362, [@weekface](https://github.com/weekface))
- Update TiDB/TiKV/PD configuration to 4.0.0 GA version (#2571, [@Yisaer](https://github.com/Yi
- TiDB Operator will not do failover for PD pods which are not desired (#2570, [@Yisaer](https://github.com/Yisaer))

12.1.17 TiDB Operator 1.1 RC.4 Release Notes

Release date: May 15, 2020

TiDB Operator version: 1.1.0-rc.4

12.1.17.1 Action Required

- Separate TiDB client certificates can be used for each component. Users should migrate the old TLS configs of Backup and Restore to the new configs. Refer to #2403 for more details (#2403, [@weekface](https://github.com/weekface))

12.1.17.2 Other Notable Changes

- Fix the bug that the service annotations would be exposed in `TidbCluster` specification (#2471, [@Yisaer](https://github.com/Yisaer))
- Fix a bug when reconciling TiDB service while the `healthCheckNodePort` is already generated by Kubernetes (#2438, [@aylei](https://github.com/aylei))
- Support `TidbMonitorRef` in `TidbCluster` Status (#2424, [@Yisaer](https://github.com/Yisaer))
- Support setting the backup path prefix for remote storage (#2435, [@onlymellb](https://github.com/onlymellb))
- Support customizing `mydumper` options in Backup CR (#2407, [@onlymellb](https://github.com/only
- Support TiCDC in TidbCluster CR. (#2338, [@weekface](https://github.com/weekface))
- Update BR to `v3.1.1` in the `tidb-backup-manager` image (#2425, [@DanielZhangQD](https://githu
- Support creating node pools for TiFlash and CDC on ACK (#2420, [@DanielZhangQD](https://githu
- Support creating node pools for TiFlash and CDC on EKS (#2413, [@DanielZhangQD](https://githul
- Expose `PVReclaimPolicy` for `TidbMonitor` when storage is enabled (#2379, [@Yisaer](https://github.com/Yisaer))
- Support arbitrary topology-based HA in tidb-scheduler (e.g. node zones) (#2366, [@PengJi](https://github.com/PengJi))
- Skip setting the TLS for PD dashboard when the TiDB version is earlier than 4.0.0 (#2389, [@weekface](https://github.com/weekface))
- Support backup and restore with GCS using BR (#2267, [@shuijing198799](https://github.com/shuij
- Update `TiDBConfig` and `TiKVConfig` to support the `4.0.0-rc` version (#2322, [@Yisaer](https://github.com/Yisaer))
- Fix the bug when `TidbCluster` service type is `NodePort`, the value of `NodePort` would change frequently (#2284, [@Yisaer](https://github.com/Yisaer))
- Add external strategy ability for `TidbClusterAutoScaler` (#2279, [@Yisaer](https://github.com/Yis
- PVC will not be deleted when `TidbMonitor` gets deleted (#2374, [@Yisaer](https://github.com/Yisae
- Support scaling for TiFlash (#2237, [@DanielZhangQD](https://github.com/DanielZhangQD))

12.1.18 TiDB Operator 1.1 RC.3 Release Notes

Release date: April 30, 2020

TiDB Operator version: 1.1.0-rc.3

12.1.18.1 Notable Changes

- Skip auto-failover when pods are not scheduled and perform recovery operation no matter what state failover pods are in (#2263, [@cofyc](https://github.com/cofyc))

- Support `TiFlash` metrics in `TidbMonitor` (#2341, [@Yisaer](https://github.com/Yisaer))
- Do not print `rclone` config in the Pod logs (#2343, [@DanielZhangQD](https://github.com/DanielZh
- Using `Patch` in `periodicity` controller to avoid updating `StatefulSet` to the wrong state (#2332, [@Yisaer](https://github.com/Yisaer))
- Set `enable-placement-rules` to `true` for PD if TiFlash is enabled in the cluster (#2328, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Support `rclone` options in the Backup and Restore CR (#2318, [@DanielZhangQD](https://github.co
- Fix the issue that statefulsets are updated during each sync even if no changes are made to the config (#2308, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Support configuring `Ingress` in `TidbMonitor` (#2314, [@Yisaer](https://github.com/Yisaer))
- Fix a bug that auto-created failover pods can't be deleted when they are in the failed state (#2300, [@cofyc](https://github.com/cofyc))
- Add useful `Event` in `TidbCluster` during upgrading and scaling when `admissionWebhook` ↪ `.validation.pods` in operator configuration is enabled (#2305, [@Yisaer](https://github.com/Yi
- Fix the issue that services are updated during each sync even if no changes are made to the service configuration (#2299, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Fix a bug that would cause panic in statefulset webhook when the update strategy of `StatefulSet` is not `RollingUpdate` (#2291, [@Yisaer](https://github.com/Yisaer))
- Fix a panic in syncing `TidbClusterAutoScaler` status when the target `TidbCluster` does not exist (#2289, [@Yisaer](https://github.com/Yisaer))
- Fix the `pdapi` cache issue while the cluster TLS is enabled (#2275, [@weekface](https://github.com/weekface))
- Fix the config error in restore (#2250, [@Yisaer](https://github.com/Yisaer))
- Support failover for TiFlash (#2249, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Update the default `eks` version in terraform scripts to 1.15 (#2238, [@Yisaer](https://github.com/Yis
- Support upgrading for TiFlash (#2246, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Add `stderr` logs from BR to the backup-manager logs (#2213, [@DanielZhangQD](https://github.co
- Add field `TiKVEncryptionConfig` in `TiKVConfig`, which defines how to encrypt data key and raw data in TiKV, and how to back up and restore the master key. See the description for details in `tikv_config.go` (#2151, [@shuijing198799](https://github.com/shuijing198799))

12.1.19 TiDB Operator 1.1 RC.2 Release Notes

Release date: April 15, 2020

TiDB Operator version: 1.1.0-rc.2

12.1.19.1 Action Required

- Change TiDB pod `readiness` probe from `HTTPGet` to `TCPSocket` 4000 port. This will trigger rolling-upgrade for the `tidb-server` component. You can set `spec.paused` ↪ to `true` before upgrading tidb-operator to avoid the rolling upgrade, and set it back to `false` when you are ready to upgrade your TiDB server (#2139, [@weekface](https://github.com/weekface))

12.1.19.2 Notable Changes

- Add `status` field for `TidbAutoScaler` CR (#2182, [@Yisaer](https://github.com/Yisaer))
- Add `spec.pd.maxFailoverCount` field to limit max failover replicas for PD (#2184, [@cofyc](https://github.com/cofyc))
- Emit more events for `TidbCluster` and `TidbClusterAutoScaler` to help users know TiDB running status (#2150, [@Yisaer](https://github.com/Yisaer))
- Add the `AGE` column to show creation timestamp for all CRDs (#2168, [@cofyc](https://github.com/cofyc))
- Add a switch to skip PD Dashboard TLS configuration (#2143, [@weekface](https://github.com/week
- Support deploying TiFlash with TidbCluster CR (#2157, [@DanielZhangQD](https://github.com/Da
- Add TLS support for TiKV metrics API (#2137, [@weekface](https://github.com/weekface))
- Set PD DashboardConfig when TLS between the MySQL client and TiDB server is enabled (#2085, [@weekface](https://github.com/weekface))
- Remove unnecessary informer caches to reduce the memory footprint of tidb-controller-manager (#1504, [@aylei](https://github.com/aylei))
- Fix the failure that Helm cannot load the kubeconfig file when deleting the tidb-operator release during `terraform destroy` (#2148, [@DanielZhangQD](https://github.com/Daniel
- Support configuring the Webhook TLS setting by loading a secret (#2135, [@Yisaer](https://github.com/Yisaer))
- Support TiFlash in TidbCluster CR (#2122, [@DanielZhangQD](https://github.com/DanielZhangQI
- Fix the error that alertmanager couldn't be set in `TidbMonitor` (#2108, [@Yisaer](https://github.com

12.1.20 TiDB Operator 1.1 RC.1 Release Notes

Release date: April 1, 2020

TiDB Operator version: 1.1.0-rc.1

12.1.20.1 Action Required

- `--advertise-address` will be configured for `tidb-server`, which would trigger rolling-upgrade for the `tidb-server` component. You can set `spec.paused` to `true` before upgrading tidb-operator to avoid the rolling upgrade, and set it back to `false` when you are ready to upgrade your TiDB server (#2076, [@cofyc](https://github.com/cofyc))
- Add the `tlsClient.tlsSecret` field in the backup and restore spec, which supports specifying a secret name that includes the cert (#2003, [@shuijing198799](https://github.com/shuijing
- Remove `spec.br.pd`, `spec.br.ca`, `spec.br.cert`, `spec.br.key` and add `spec.br.cluster`, `spec.br.clusterNamespace` for the `Backup`, `Restore` and `BackupSchedule` custom resources, which makes the BR configuration more reasonable (#1836, [@shuijing198799](https://github.com/shuijing198799))

12.1.20.2 Other Notable Changes

- Use `tidb-lightning` in `Restore` instead of `loader` (#2068, [@Yisaer](https://github.com/Yisaer))
- Add `cert-allowed-cn` support to TiDB components (#2061, [@weekface](https://github.com/weekf
- Fix the PD `location-labels` configuration (#1941, [@aylei](https://github.com/aylei))
- Able to pause and unpause TiDB cluster deployment via `spec.paused` (#2013, [@cofyc](https://github.com/cofyc))
- Default the `max-backups` for TiDB server configuration to `3` if the TiDB cluster is deployed by CR (#2045, [@Yisaer](https://github.com/Yisaer))
- Able to configure custom environments for components (#2052, [@cofyc](https://github.com/cofyc))
- Fix the error that `kubectl get tc` cannot show correct images (#2031, [@Yisaer](https://github.com

    1. Default the `spec.tikv.maxFailoverCount` and `spec.tidb.maxFailoverCount` to `3` when they are not defined
    2. Disable auto-failover when `maxFailoverCount` is set to `0` (#2015, [@Yisaer](https://github.com/

- Support deploying TiDB clusters with TidbCluster and TidbMonitor CRs via Terraform on ACK (#2012, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Update PDConfig for TidbCluster to PD v3.1.0 (#1928, [@Yisaer](https://github.com/Yisaer))
- Support deploying TiDB clusters with TidbCluster and TidbMonitor CRs via Terraform on AWS (#2004, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Update TidbConfig for TidbCluster to TiDB v3.1.0 (#1906, [@Yisaer](https://github.com/Yisaer))
- Allow users to define resources for initContainers in TiDB initializer job (#1938, [@tfulcrand](https://github.com/tfulcrand))
- Add TLS support for Pump and Drainer (#1979, [@weekface](https://github.com/weekface))
- Add documents and examples for auto-scaler and initializer (#1772, [@Yisaer](https://github.com/Yi

    1. Add check to guarantee the NodePort won't be changed if the serviceType of TidbMonitor is NodePort
    2. Add EnvVar sort to avoid the monitor rendering different results from the same TidbMonitor spec
    3. Fix the problem that the TidbMonitor LoadBalancer IP is not used (#1962, [@Yisaer](https://github.com/Yisaer))

- Make tidb-initializer support TLS (#1931, [@weekface](https://github.com/weekface))

    1. Fix the problem that Advanced StatefulSet cannot work with webhook
    2. Change the Reaction for the Down State TiKV pod during deleting request in webhook from admit to reject (#1963, [@Yisaer](https://github.com/Yisaer))

- Fix the drainer installation error when `drainerName` is set (#1961, [@DanielZhangQD](https://github
- Fix some TiKV configuration keys in toml (#1887, [@aylei](https://github.com/aylei))
- Support using a remote directory as data source for tidb-lightning (#1629, [@aylei](https://github.com/aylei))
- Add the API document and a script that generates documentation (#1945, [@Yisaer](https://github.com/Yisaer))
- Add the tikv-importer chart (#1910, [@shonge](https://github.com/shonge))
- Fix the Prometheus scrape config issue while TLS is enabled (#1919, [@weekface](https://github.com/weekface))
- Enable TLS between TiDB components (#1870, [@weekface](https://github.com/weekface))

- Fix the timeout error when `.Values.admission.validation.pods` is `true` during the TiKV upgrade (#1875, [@Yisaer](https://github.com/Yisaer))
- Enable TLS for MySQL clients (#1878, [@weekface](https://github.com/weekface))
- Fix the bug which would cause broken TiDB image property (#1860, [@Yisaer](https://github.com/Y
- TidbMonitor would use its namespace for the targetRef if it is not defined (#1834, [@Yisaer](https://github.com/Yisaer))
- Support starting tidb-server with `--advertise-address` parameter (#1859, [@LinuxGit](https://github.com/LinuxGit))
- Backup/Restore: support configuring TiKV GC life time (#1835, [@LinuxGit](https://github.com/Li
- Support no secret for S3/Ceph when the OIDC authentication is used (#1817, [@tirsen](https://github.com/tirsen))

    1. Change the setting from the previous `admission.hookEnabled.pods` to the `admission.validation.pods`
    2. Change the setting from the previous `admission.hookEnabled.statefulSets` to the `admission.validation.statefulSets`
    3. Change the setting from the previous `admission.hookEnabled.validating` to the `admission.validation.pingcapResources`
    4. Change the setting from the previous `admission.hookEnabled.defaulting` to the `admission.mutation.pingcapResources`
    5. Change the setting from the previous `admission.failurePolicy.defaulting` to the `admission.failurePolicy.mutation`
    6. Change the setting from the previous `admission.failurePolicy.*` to the `admission.failurePolicy.validation` (#1832, [@Yisaer](https://github.com/Yisaer))

- Enable TidbCluster defaulting mutation by default which is recommended when admission webhook is used (#1816, [@Yisaer](https://github.com/Yisaer))
- Fix a bug that TiKV fails to start while creating the cluster using CR with cluster TLS enabled (#1808, [@weekface](https://github.com/weekface))
- Support using prefix in remote storage during backup/restore (#1790, [@DanielZhangQD](https://git

12.1.21 TiDB Operator 1.1 Beta.2 Release Notes

Release date: February 26, 2020

TiDB Operator version: 1.1.0-beta.2

12.1.21.1 Action Required

- `--default-storage-class-name` and `--default-backup-storage-class-name` are abandoned, and the storage class defaults to Kubernetes default storage class right now. If you have set default storage class different than Kubernetes default storage class, please set them explicitly in your TiDB cluster helm or YAML files. (#1581, [@cofyc](https://github.com/cofyc))

12.1.21.2 Other Notable Changes

- Allow users to configure affinity and tolerations for `Backup` and `Restore`. (#1737, [@Smana](https://github.com/Smana))
- Allow AdvancedStatefulSet and Admission Webhook to work together. (#1640, [@Yisaer](https://github.com/Yisaer))
- Add a basic deployment example of managing TiDB cluster with custom resources only. (#1573, [@aylei](https://github.com/aylei))
- Support TidbCluster Auto-scaling feature based on CPU average utilization load. (#1731, [@Yisaer](https://github.com/Yisaer))
- Support user-defined TiDB server/client certificate (#1714, [@weekface](https://github.com/weekfac
- Add an option for tidb-backup chart to allow reusing existing PVC or not for restore (#1708, [@mightyguava](https://github.com/mightyguava))
- Add `resources`, `imagePullPolicy` and `nodeSelector` field for tidb-backup chart (#1705, [@mightyguava](https://github.com/mightyguava))
- Add more SANs (Subject Alternative Name) to TiDB server certificate (#1702, [@weekface](https://github.com/weekface))
- Support automatically migrating existing Kubernetes StatefulSets to Advanced StatefulSets when AdvancedStatfulSet feature is enabled (#1580, [@cofyc](https://github.com/cofyc))
- Fix the bug in admission webhook which causes PD pod deleting error and allow the deleting pod to request for PD and TiKV when PVC is not found. (#1568, [@Yisaer](https://github.com/Yisaer))
- Limit the restart rate for PD and TiKV - only one instance would be restarted each time (#1532, [@Yisaer](https://github.com/Yisaer))
- Add default ClusterRef namespace for TidbMonitor as the same as it is deployed and fix the bug that TidbMonitor's Pod can't be created when Spec.PrometheusSpec.logLevel is missing. (#1500, [@Yisaer](https://github.com/Yisaer))
- Refine logs for `TidbMonitor` and `TidbInitializer` controller (#1493, [@aylei](https://github.com/ay
- Avoid unnecessary updates to `Service` and `Deployment` of discovery (#1499, [@aylei](https://github.com/aylei))
- Remove some update events that are not very useful (#1486, [@weekface](https://github.com/weekfac

12.1.22 TiDB Operator 1.1 Beta.1 Release Notes

Release date: January 8, 2020

TiDB Operator version: 1.1.0-beta.1

12.1.22.1 Action Required

- ACTION REQUIRED: Add the `timezone` support for all charts (#1122, [@weekface](https://github.com/weekface)).

  For the `tidb-cluster` chart, we already have the `timezone` option (`UTC` by default). If the user does not change it to a different value (for example: `Aisa/Shanghai`), all Pods will not be recreated.

If the user changes it to another value (for example: `Aisa/Shanghai`), all the related Pods (add a `TZ` env) will be recreated (rolling update).

Regarding other charts, we don't have a `timezone` option in their `values.yaml`. We add the `timezone` option in this PR. No matter whether the user uses the old `values.yaml` or the new `values.yaml`, all the related Pods (add a `TZ` env) will not be recreated (rolling update).

The related Pods include `pump`, `drainer`, `discovery`, `monitor`, `scheduled backup`, `tidb-initializer`, and `tikv-importer`.

All images' time zone maintained by `tidb-operator` is `UTC`. If you use your own images, you need to make sure that the time zone inside your images is `UTC`.

12.1.22.2 Other Notable Changes

- Support backup to S3 with Backup & Restore (BR) (#1280, [@DanielZhangQD](https://github.com/
- Add basic defaulting and validating for `TidbCluster` (#1429, [@aylei](https://github.com/aylei))
- Support scaling in/out with deleted slots feature of advanced StatefulSets (#1361, [@cofyc](https://github.com/cofyc))
- Support initializing the TiDB cluster with TidbInitializer Custom Resource (#1403, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Refine the configuration schema of PD/TiKV/TiDB (#1411, [@aylei](https://github.com/aylei))
- Set the default name of the instance label key for `tidbcluster`-owned resources to the cluster name (#1419, [@aylei](https://github.com/aylei))
- Extend the custom resource `TidbCluster` to support managing the Pump cluster (#1269, [@aylei](https://github.com/aylei))
- Fix the default TiKV-importer configuration (#1415, [@aylei](https://github.com/aylei))
- Expose ephemeral-storage in resource configuration (#1398, [@aylei](https://github.com/aylei))
- Add e2e case of operating tidb-cluster without helm (#1396, [@aylei](https://github.com/aylei))
- Expose terraform Aliyun ACK version and specify the default version to '1.14.8-aliyun.1' (#1284, [@shonge](https://github.com/shonge))
- Refine error messages for the scheduler (#1373, [@weekface](https://github.com/weekface))
- Bind the cluster-role `system:kube-scheduler` to the service account `tidb-scheduler` (#1355, [@shonge](https://github.com/shonge))
- Add a new CRD TidbInitializer (#1391, [@aylei](https://github.com/aylei))
- Upgrade the default backup image to pingcap/tidb-cloud-backup:20191217 and facilitate the `-r` option (#1360, [@aylei](https://github.com/aylei))
- Fix Docker ulimit configuring for the latest EKS AMI (#1349, [@aylei](https://github.com/aylei))
- Support sync pump status to tidb-cluster (#1292, [@shonge](https://github.com/shonge))
- Support automatically creating and reconciling the tidb-discovery-service for `tidb-controller-manager` (#1322, [@aylei](https://github.com/aylei))
- Make backup and restore more universal and secure (#1276, [@onlymellb](https://github.com/onlyme
- Manage PD and TiKV configurations in the `TidbCluster` resource (#1330, [@aylei](https://github.com/aylei))
- Support managing the configuration of tidb-server in the `TidbCluster` resource (#1291, [@aylei](https://github.com/aylei))

- Add schema for configuration of TiKV (#1306, [@aylei](https://github.com/aylei))
- Wait for the TiDB `host:port` to be opened before processing to initialize TiDB to speed up TiDB initialization (#1296, [@cofyc](https://github.com/cofyc))
- Remove DinD related scripts (#1283, [@shonge](https://github.com/shonge))
- Allow retrieving credentials from metadata on AWS and GCP (#1248, [@gregwebs](https://github.com/gregwebs))
- Add the privilege to operate configmap for tidb-controller-manager (#1275, [@aylei](https://github.com/aylei))
- Manage TiDB service in tidb-controller-manager (#1242, [@aylei](https://github.com/aylei))
- Support the cluster-level setting for components (#1193, [@aylei](https://github.com/aylei))
- Get the time string from the current time instead of the Pod name (#1229, [@weekface](https://github.com/weekface))
- Operator will not resign the ddl owner anymore when upgrading tidb-servers because tidb-server will transfer ddl owner automatically on shutdown (#1239, [@aylei](https://github.com/aylei))
- Fix the Google terraform module `use_ip_aliases` error (#1206, [@tennix](https://github.com/tenni
- Upgrade the default TiDB version to v3.0.5 (#1179, [@shonge](https://github.com/shonge))
- Upgrade the base system of Docker images to the latest stable (#1178, [@AstroProfundis](https://github.com/AstroProfundis))
- `tkctl get TiKV` now can show store state for each TiKV Pod (#916, [@Yisaer](https://github.com/Y
- Add an option to monitor across namespaces (#907, [@gregwebs](https://github.com/gregwebs))
- Add the `STOREID` column to show the store ID for each TiKV Pod in `tkctl get TiKV` (#842, [@Yisaer](https://github.com/Yisaer))
- Users can designate permitting host in chart values.tidb.permitHost (#779, [@shonge](https://github.com/shonge))
- Add the zone label and reserved resources arguments to kubelet (#871, [@aylei](https://github.com/a
- Fix an issue that kubeconfig may be destroyed in the apply phrase (#861, [@cofyc](https://github.com/cofyc))
- Support canary release for the TiKV component (#869, [@onlymellb](https://github.com/onlymellb)
- Make the latest charts compatible with the old controller manager (#856, [@onlymellb](https://github.com/onlymellb))
- Add the basic support of TLS encrypted connections in the TiDB cluster (#750, [@AstroProfundis](https://github.com/AstroProfundis))
- Support tidb-operator to spec nodeSelector, affinity and tolerations (#855, [@shonge](https://github.com/shonge))
- Support configuring resources requests and limits for all containers of the TiDB cluster (#853, [@aylei](https://github.com/aylei))
- Support using Kind (Kubernetes IN Docker) to set up a testing environment (#791, [@xiaojingchen](https://github.com/xiaojingchen))
- Support ad-hoc data source to be restored with the tidb-lightning chart (#827, [@tennix](https://github.com/tennix))
- Add the `tikvGCLifeTime` option (#835, [@weekface](https://github.com/weekface))
- Update the default backup image to pingcap/tidb-cloud-backup:20190828 (#846, [@aylei](https://github.com/aylei))
- Fix the Pump/Drainer data directory to avoid potential data loss (#826, [@aylei](https://github.com/

- Fix the issue that`tkctl` ouputs nothing with the `-oyaml` or `-ojson` flag and support viewing details of a specific Pod or PV, also improve the output of the `tkctl get` command (#822, [@onlymellb](https://github.com/onlymellb))
- Add recommendations options to mydumper: `-t 16 -F 64 --skip-tz-utc` (#828, [@weekface](https://github.com/weekface))
- Support zonal and multi-zonal clusters in deploy/gcp (#809, [@cofyc](https://github.com/cofyc))
- Fix ad-hoc backup when the default backup name is used (#836, [@DanielZhangQD](https://github.c
- Add the support for tidb-lightning (#817, [@tennix](https://github.com/tennix))
- Support restoring the TiDB cluster from a specified scheduled backup directory (#804, [@onlymellb](https://github.com/onlymellb))
- Fix an exception in the log of `tkctl` (#797, [@onlymellb](https://github.com/onlymellb))
- Add the `hostNetwork` field in PD/TiKV/TiDB spec to make it possible to run TiDB components in host network (#774, [@cofyc](https://github.com/cofyc))
- Use mdadm and RAID rather than LVM when it is available on GKE (#789, [@gregwebs](https://github.com/gregwebs))
- Users can now expand cloud storage PV dynamically by increasing the PVC storage size (#772, [@tennix](https://github.com/tennix))
- Support configuring node image types for PD/TiDB/TiKV node pools (#776, [@cofyc](https://github.com/cofyc))
- Add a script to delete unused disk for GKE (#771, [@gregwebs](https://github.com/gregwebs))
- Support `binlog.pump.config` and `binlog.drainer.config` configurations for Pump and Drainer (#693, [@weekface](https://github.com/weekface))
- Prevent the Pump progress from exiting with 0 if the Pump becomes `offline` (#769, [@weekface](https://github.com/weekface))
- Introduce a new helm chart, tidb-drainer, to facilitate multiple Drainers management (#744, [@aylei](https://github.com/aylei))
- Add the backup-manager tool to support backing up, restoring, and cleaning backup data (#694, [@onlymellb](https://github.com/onlymellb))
- Add `affinity` to Pump/Drainer configration (#741, [@weekface](https://github.com/weekface))
- Fix the TiKV scaling failure in some cases after TiKV failover (#726, [@onlymellb](https://github.com/onlymellb))
- Fix error handling for UpdateService (#718, [@DanielZhangQD](https://github.com/DanielZhangQ
- Reduce e2e run time from 60 m to 20 m (#713, [@weekface](https://github.com/weekface))
- Add the `AdvancedStatefulset` feature to use advanced StatefulSet instead of Kubernetes builtin StatefulSet (#1108, [@cofyc](https://github.com/cofyc))
- Enable auto generate certificates for the TiDB cluster (#782, [@AstroProfundis](https://github.com/AstroProfundis))
- Support backup to gcs (#1127, [@onlymellb](https://github.com/onlymellb))
- Support configuring `net.ipv4.tcp_keepalive_time` and `net.core.somaxconn` for TiDB and configuring `net.core.somaxconn` for TiKV (#1107, [@DanielZhangQD](https://github.co
- Add basic e2e tests for aggregated apiserver (#1109, [@aylei](https://github.com/aylei))
- Add the `enablePVReclaim` option to reclaim PV when tidb-operator scales in TiKV or PD (#1037, [@onlymellb](https://github.com/onlymellb))
- Unify all S3 compliant storage to support backup and restore (#1088, [@onlymellb](https://github.com/onlymellb))

- Set podSecuriyContext to nil by default (#1079, [@aylei](https://github.com/aylei))
- Add tidb-apiserver in the tidb-operator chart (#1083, [@aylei](https://github.com/aylei))
- Add new component TiDB aggregated apiserver (#1048, [@aylei](https://github.com/aylei))
- Fix the issue that the tkctl version does not work when the release name is un-wanted (#1065, [@aylei](https://github.com/aylei))
- Support pause for backup schedule (#1047, [@onlymellb](https://github.com/onlymellb))
- Fix the issue that TiDB Loadbalancer is empty in terraform output (#1045, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Fix that the `create_tidb_cluster_release` variable in AWS terraform script does not work (#1062, [@aylei](https://github.com/aylei))
- Enable `ConfigMapRollout` by default in the stability test (#1036, [@aylei](https://github.com/aylei))
- Migrate to use app/v1 and do not support Kubernetes before 1.9 anymore (#1012, [@Yisaer](https://github.com/Yisaer))
- Suspend the ReplaceUnhealthy process for AWS TiKV auto-scaling-group (#1014, [@aylei](https://github.com/aylei))
- Change the tidb-monitor-reloader image to pingcap/tidb-monitor-reloader:v1.0.1 (#898, [@qiffang](https://github.com/qiffang))
- Add some sysctl kernel parameter settings for tuning (#1016, [@tennix](https://github.com/tennix))
- Support maximum retention time backups for backup schedule (#979, [@onlymellb](https://github.com/onlymellb))
- Upgrade the default TiDB version to v3.0.4 (#837, [@shonge](https://github.com/shonge))
- Fix values file customization for tidb-operator on Aliyun (#971, [@DanielZhangQD](https://github.co
- Add the `maxFailoverCount` limit to TiKV (#965, [@weekface](https://github.com/weekface))
- Support setting custom tidb-operator values in terraform script for AWS (#946, [@aylei](https://github.com/aylei))
- Convert the TiKV capacity into MiB when it is not a multiple of GiB (#942, [@cofyc](https://github.com/cofyc))
- Fix Drainer misconfiguration (#939, [@weekface](https://github.com/weekface))
- Support correctly deploying tidb-operator and tidb-cluster with customized `values.yaml` (#959, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Support specifying SecurityContext for PD, TiKV and TiDB Pods and enable tcp keepalive for AWS (#915, [@aylei](https://github.com/aylei))

12.2 v1.0

12.2.1 TiDB Operator 1.0.7 Release Notes

Release date: June 16, 2020

TiDB Operator version: 1.0.7

12.2.1.1 Notable Changes

- Fix alert rules lost after rolling upgrade (#2715)

- Upgrade local volume provisioner to 2.3.4 (#1778)
- Fix operator failover config invalid (#1877)
- Remove unnecessary duplicated docs (#2100)
- Update doc links and image in readme (#2106)
- Emit events when PD failover (#1466)
- Fix some broken urls (#1501)
- Remove some not very useful update events (#1486)

12.2.2 TiDB Operator 1.0.6 Release Notes

Release date: December 27, 2019

TiDB Operator version: 1.0.6

12.2.2.1 v1.0.6 What's New

Action required: Users should migrate the configs in `values.yaml` of previous chart releases to the new `values.yaml` of the new chart. Otherwise, the monitor pods might fail when you upgrade the monitor with the new chart.

For example, configs in the old `values.yaml` file:

```
monitor:
  ...
  initializer:
    image: pingcap/tidb-monitor-initializer:v3.0.5
    imagePullPolicy: IfNotPresent
  ...
```

After migration, configs in the new `values.yaml` file should be as follows:

```
monitor:
  ...
  initializer:
    image: pingcap/tidb-monitor-initializer:v3.0.5
    imagePullPolicy: Always
    config:
      K8S_PROMETHEUS_URL: http://prometheus-k8s.monitoring.svc:9090
  ...
```

12.2.2.1.1 Monitor

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

12.2.2.1.2 TiDB Scheduler

- Refine scheduler error messages (#1373)

12.2.2.1.3 Compatibility

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

12.2.2.1.4 TiKV Importer

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

12.2.2.1.5 E2E

- Ensure pods unaffected when upgrading (#955)

12.2.2.1.6 CI

- Move the release CI script from Jenkins into the tidb-operator repository (#1237)
- Adjust the release CI script for the `release-1.0` branch (#1320)

12.2.3 TiDB Operator 1.0.5 Release Notes

Release date: December 11, 2019

TiDB Operator version: 1.0.5

12.2.3.1 v1.0.5 What's New

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

12.2.3.1.1 Scheduled Backup

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

12.2.3.1.2 TiDB Binlog

- It is recommended that TiDB and Pump be deployed on the same node through the `affinity` feature and Pump be dispersed on different nodes through the `anti-affinity` feature. At most only one Pump instance is allowed on each node. We added a guide to the chart. (#1251)

12.2.3.1.3 Compatibility

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

12.2.3.1.4 E2E

- Fix e2e nil point dereference (#1221)

12.2.4 TiDB Operator 1.0.4 Release Notes

Release date: November 23, 2019

TiDB Operator version: 1.0.4

12.2.4.1 v1.0.4 What's New

12.2.4.1.1 Action Required

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

12.2.4.1.2 Highlights

#1202 introduced `HostNetwork` support, which offers better performance compared to the Pod network. Check out our benchmark report for details.

> **Note:**
>
> Due to this issue of Kubernetes, the Kubernetes cluster must be one of the following versions to enable `HostNetwork` of the TiDB cluster:
>
> - `v1.13.11` or later
> - `v1.14.7` or later
> - `v1.15.4` or later
> - any version since `v1.16.0`

#1175 added the `podSecurityContext` support for TiDB cluster Pods. We recommend setting the namespaced kernel parameters for TiDB cluster Pods according to our Environment Recommendation.

New Helm chart `tidb-lightning` brings TiDB Lightning support for TiDB in Kubernetes. Check out the document for detailed user guide.

Another new Helm chart `tidb-drainer` brings multiple drainers support for TiDB Binlog in Kubernetes. Check out the document for detailed user guide.

12.2.4.1.3 Improvements

- Support HostNetwork (#1202)
- Support configuring sysctls for Pods and enable net.* (#1175)
- Add tidb-lightning support (#1161)
- Add new helm chart tidb-drainer to support multiple drainers (#1160)

12.2.4.2 Detailed Bug Fixes and Changes

- Add e2e scripts and simplify the e2e Jenkins file (#1174)
- Fix the pump/drainer data directory to avoid data loss caused by bad configuration (#1183)
- Add init sql case to e2e (#1199)
- Keep the instance label of drainer same with the TiDB cluster in favor of monitoring (#1170)
- Set `podSecuriyContext` to nil by default in favor of backward compatibility (#1184)

12.2.4.3 Additional Notes for Users of v1.1.0.alpha branch

For historical reasons, `v1.1.0.alpha` is a hot-fix branch and got this name by mistake. All fixes in that branch are cherry-picked to `v1.0.4` and the `v1.1.0.alpha` branch will be discarded to keep things clear.

We strongly recommend you to upgrade to `v1.0.4` if you are using any version under `v1.1.0.alpha`.

`v1.0.4` introduces the following fixes comparing to `v1.1.0.alpha.3`:

- Support HostNetwork (#1202)
- Add the permit host option for tidb-initializer job (#779)
- Fix drainer misconfiguration in tidb-cluster chart (#945)
- Set the default `externalTrafficPolicy` to be Local for TiDB services (#960)
- Fix tidb-operator crash when users modify sts upgrade strategy improperly (#969)
- Add the `maxFailoverCount` limit to TiKV (#976)
- Fix values file customization for tidb-operator on Aliyun (#983)
- Do not limit failover count when maxFailoverCount = 0 (#978)
- Suspend the `ReplaceUnhealthy` process for TiKV auto-scaling-group on AWS (#1027)
- Fix the issue that the `create_tidb_cluster_release` variable does not work (#1066))
- Add `v1` to statefulset apiVersions (#1056)
- Add timezone support (#1126)

12.2.5 TiDB Operator 1.0.3 Release Notes

Release date: November 13, 2019

TiDB Operator version: 1.0.3

12.2.5.1 v1.0.3 What's New

12.2.5.1.1 Action Required

ACTION REQUIRED: This release upgrades default TiDB version to `v3.0.5` which fixed a serious bug in TiDB. So if you are using TiDB `v3.0.4` or prior versions, you **must** upgrade to `v3.0.5`.

ACTION REQUIRED: This release adds the `timezone` support for all charts.

For existing TiDB clusters. If the `timezone` in `tidb-cluster/values.yaml` has been customized to other timezones instead of the default `UTC`, then upgrading tidb-operator will trigger a rolling update for the related pods.

The related pods include `pump`, `drainer`, `discovery`, `monitor`, `scheduled backup`, `tidb-initializer`, and `tikv-importer`.

The time zone for all images maintained by `tidb-operator` should be `UTC`. If you use your own images, you need to make sure that the corresponding time zones are `UTC`.

12.2.5.1.2 Improvements

- Add the `timezone` support for all containers of the TiDB cluster
- Support configuring resource requests and limits for all containers of the TiDB cluster

12.2.5.2 Detailed Bug Fixes and Changes

- Upgrade default TiDB version to `v3.0.5` (#1132)
- Add the `timezone` support for all containers of the TiDB cluster (#1122)
- Support configuring resource requests and limits for all containers of the TiDB cluster (#853)

12.2.6 TiDB Operator 1.0.2 Release Notes

Release date: November 1, 2019

TiDB Operator version: 1.0.2

12.2.6.1 v1.0.2 What's New

12.2.6.1.1 Action Required

The AWS Terraform script uses auto-scaling-group for all components (PD/TiKV/TiDB/monitor). When an ec2 instance fails the health check, the instance will be replaced. This is helpful for those applications that are stateless or use EBS volumes to store data.

But a TiKV Pod uses instance store to store its data. When an instance is replaced, all the data on its store will be lost. TiKV has to resync all data to the newly added instance. Though TiDB is a distributed database and can work when a node fails, resyncing data can cost much if the dataset is large. Besides, the ec2 instance may be recovered to a healthy state by rebooting.

So we disabled the auto-scaling-group's replacing behavior in `v1.0.2`.

Auto-scaling-group scaling process can also be suspended according to its documentation if you are using `v1.0.1` or prior versions.

12.2.6.1.2 Improvements

- Suspend ReplaceUnhealthy process for AWS TiKV auto-scaling-group
- Add a new VM manager `qm` in stability test
- Add `tikv.maxFailoverCount` limit to TiKV
- Set the default `externalTrafficPolicy` to be `Local` for TiDB service in AWS/GCP/Aliyun
- Add provider and module versions for AWS

12.2.6.1.3 Bug Fixes

- Fix the issue that tkctl version does not work when the release name is un-wanted
- Migrate statefulsets apiVersion to `app/v1` which fixes compatibility with Kubernetes 1.16 and above versions
- Fix the issue that the `create_tidb_cluster_release` variable in AWS Terraform script does not work
- Fix compatibility issues by adding `v1beta1` to statefulset apiVersions
- Fix the issue that TiDB Loadbalancer is empty in Terraform output
- Fix a compatibility issue of TiKV `maxFailoverCount`
- Fix Terraform providers version constraint issues for GCP and Aliyun
- Fix values file customization for tidb-operator on Aliyun
- Fix tidb-operator crash when users modify statefulset upgrade strategy improperly
- Fix drainer misconfiguration

12.2.6.2 Detailed Bug Fixes and Changes

- Fix the issue that tkctl version does not work when the release name is un-wanted (#1065)
- Fix the issue that the `create_tidb_cluster_release` variable in AWS terraform script does not work (#1062)

- Fix compatibility issues for (#1012): add `v1beta1` to statefulset apiVersions (#1054)
- Enable ConfigMapRollout by default in stability test (#1036)
- Fix the issue that TiDB Loadbalancer is empty in Terraform output (#1045)
- Migrate statefulsets apiVersion to `app/v1` which fixes compatibility with Kubernetes 1.16 and above versions (#1012)
- Only expect TiDB cluster upgrade to be complete when rolling back wrong configuration in stability test (#1030)
- Suspend ReplaceUnhealthy process for AWS TiKV auto-scaling-group (#1014)
- Add a new VM manager `qm` in stability test (#896)
- Fix provider versions constraint issues for GCP and Aliyun (#959)
- Fix values file customization for tidb-operator on Aliyun (#971)
- Fix a compatibility issue of TiKV `tikv.maxFailoverCount` (#977)
- Add `tikv.maxFailoverCount` limit to TiKV (#965)
- Fix tidb-operator crash when users modify statefulset upgrade strategy improperly (#912)
- Set the default `externalTrafficPolicy` to be `Local` for TiDB service in AWS/GCP/Aliyun (#947)
- Add note about setting PV reclaim policy to retain (#911)
- Fix drainer misconfiguration (#939)
- Add provider and module versions for AWS (#926)

12.2.7 TiDB Operator 1.0.1 Release Notes

Release date: September 17, 2019

TiDB Operator version: 1.0.1

12.2.7.1 v1.0.1 What's New

12.2.7.1.1 Action Required

- ACTION REQUIRED: We fixed a serious bug (#878) that could cause all `PD` and `TiKV` pods to be accidentally deleted when `kube-apiserver` fails. This would cause TiDB service outage. So if you are using `v1.0.0` or prior versions, you **must** upgrade to `v1.0.1`.
- ACTION REQUIRED: The backup tool image pingcap/tidb-cloud-backup uses a forked version of `Mydumper`. The current version `pingcap/tidb-cloud-backup :20190610` contains a serious bug that could result in a missing column in the exported data. This is fixed in #29. And the default image used now contains this fixed version. So if you are using the old version image for backup, you **must** upgrade to use `pingcap/tidb-cloud-backup:201908028` and do a new full backup to avoid potential data inconsistency.

12.2.7.1.2 Improvements

- Modularize GCP Terraform
- Add a script to remove orphaned k8s disks
- Support `binlog.pump.config`, `binlog.drainer.config` configurations for Pump and Drainer
- Set the resource limit for the `tidb-backup` job
- Add `affinity` to Pump and Drainer configurations
- Upgrade local-volume-provisioner to `v2.3.2`
- Reduce e2e run time from `60m` to `20m`
- Prevent the Pump process from exiting with `0` if the Pump becomes `offline`
- Support expanding cloud storage PV dynamically by increasing PVC storage size
- Add the `tikvGCLifeTime` option to do backup
- Add important parameters to `tikv.config` and `tidb.config` in `values.yaml`
- Support restoring the TiDB cluster from a specified scheduled backup directory
- Enable cloud storage volume expansion & label local volume
- Document and improve HA algorithm
- Support specifying the permit host in the `values.tidb.permitHost` chart
- Add the zone label and reserved resources arguments to kubelet
- Update the default backup image to `pingcap/tidb-cloud-backup:20190828`

12.2.7.1.3 Bug Fixes

- Fix the TiKV scale-in failure in some cases after the TiKV failover
- Fix error handling for UpdateService
- Fix some orphan pods cleaner bugs
- Fix the bug of setting the `StatefulSet` partition
- Fix ad-hoc full backup failure due to incorrect `claimName`
- Fix the offline Pump: the Pump process will exit with `0` if going offline
- Fix an incorrect condition judgment

12.2.7.2 Detailed Bug Fixes and Changes

- Clean up `tidb.pingcap.com/pod-scheduling` annotation when the pod is scheduled (#790)
- Update tidb-cloud-backup image tag (#846)
- Add the TiDB permit host option (#779)
- Add the zone label and reserved resources for nodes (#871)
- Fix some orphan pods cleaner bugs (#878)
- Fix the bug of setting the `StatefulSet` partition (#830)
- Add the `tikvGCLifeTime` option (#835)
- Add recommendations options to Mydumper (#828)
- Fix ad-hoc full backup failure due to incorrect `claimName` (#836)

- Improve `tkctl get` command output (#822)
- Add important parameters to TiKV and TiDB configurations (#786)
- Fix the issue that `binlog.drainer.config` is not supported in v1.0.0 (#775)
- Support restoring the TiDB cluster from a specified scheduled backup directory (#804)
- Fix `extraLabels` description in `values.yaml` (#763)
- Fix tkctl log output exception (#797)
- Add a script to remove orphaned K8s disks (#745)
- Enable cloud storage volume expansion & label local volume (#772)
- Prevent the Pump process from exiting with `0` if the Pump becomes `offline` (#769)
- Modularize GCP Terraform (#717)
- Support `binlog.pump.config` configurations for Pump and Drainer (#693)
- Remove duplicate key values (#758)
- Fix some typos (#738)
- Extend the waiting time of the `CheckManualPauseTiDB` process (#752)
- Set the resource limit for the `tidb-backup` job (#729)
- Fix e2e test compatible with v1.0.0 (#757)
- Make incremental backup test work (#764)
- Add retry logic for `LabelNodes` function (#735)
- Fix the TiKV scale-in failure in some cases (#726)
- Add affinity to Pump and Drainer (#741)
- Refine cleanup logic (#719)
- Inject a failure by pod annotation (#716)
- Update README links to point to correct `pingcap.com/docs` URLs for English and Chinese (#732)
- Document and improve HA algorithm (#670)
- Fix an incorrect condition judgment (#718)
- Upgrade local-volume-provisioner to v2.3.2 (#696)
- Reduce e2e test run time (#713)
- Fix Terraform GKE scale-out issues (#711)
- Update wording and fix format for v1.0.0 (#709)
- Update documents (#705)

12.2.8 TiDB Operator 1.0 GA Release Notes

Release date: July 30, 2019

TiDB Operator version: 1.0.0

12.2.8.1 v1.0.0 What's New

12.2.8.1.1 Action Required

- ACTION REQUIRED: `tikv.storeLabels` was removed from `values.yaml`. You can directly set it with `location-labels` in `pd.config`.

- ACTION REQUIRED: the `--features` flag of tidb-scheduler has been updated to the `key={true,false}` format. You can enable the feature by appending `=true`.
- ACTION REQUIRED: you need to change the configurations in `values.yaml` of previous chart releases to the new `values.yaml` of the new chart. Otherwise, the configurations will be ignored when upgrading the TiDB cluster with the new chart.

The `pd` section in old `values.yaml`:

```
pd:
  logLevel: info
  maxStoreDownTime: 30m
  maxReplicas: 3
```

The `pd` section in new `values.yaml`:

```
pd:
  config: |
    [log]
    level = "info"
    [schedule]
    max-store-down-time = "30m"
    [replication]
    max-replicas = 3
```

The `tikv` section in old `values.yaml`:

```
tikv:
  logLevel: info
  syncLog: true
  readpoolStorageConcurrency: 4
  readpoolCoprocessorConcurrency: 8
  storageSchedulerWorkerPoolSize: 4
```

The `tikv` section in new `values.yaml`:

```
tikv:
  config: |
    log-level = "info"
    [server]
    status-addr = "0.0.0.0:20180"
    [raftstore]
    sync-log = true
    [readpool.storage]
    high-concurrency = 4
    normal-concurrency = 4
    low-concurrency = 4
    [readpool.coprocessor]
```

```
    high-concurrency = 8
    normal-concurrency = 8
    low-concurrency = 8
    [storage]
    scheduler-worker-pool-size = 4
```

The `tidb` section in old `values.yaml`:

```
tidb:
  logLevel: info
  preparedPlanCacheEnabled: false
  preparedPlanCacheCapacity: 100
  txnLocalLatchesEnabled: false
  txnLocalLatchesCapacity: "10240000"
  tokenLimit: "1000"
  memQuotaQuery: "34359738368"
  txnEntryCountLimit: "300000"
  txnTotalSizeLimit: "104857600"
  checkMb4ValueInUtf8: true
  treatOldVersionUtf8AsUtf8mb4: true
  lease: 45s
  maxProcs: 0
```

The `tidb` section in new `values.yaml`:

```
tidb:
  config: |
    token-limit = 1000
    mem-quota-query = 34359738368
    check-mb4-value-in-utf8 = true
    treat-old-version-utf8-as-utf8mb4 = true
    lease = "45s"
    [log]
    level = "info"
    [prepared-plan-cache]
    enabled = false
    capacity = 100
    [txn-local-latches]
    enabled = false
    capacity = 10240000
    [performance]
    txn-entry-count-limit = 300000
    txn-total-size-limit = 104857600
    max-procs = 0
```

The `monitor` section in old `values.yaml`:

```
monitor:
  create: true
  ...
```

The `monitor` section in new `values.yaml`:

```
monitor:
  create: true
   initializer:
     image: pingcap/tidb-monitor-initializer:v3.0.5
     imagePullPolicy: IfNotPresent
   reloader:
     create: true
     image: pingcap/tidb-monitor-reloader:v1.0.0
     imagePullPolicy: IfNotPresent
     service:
       type: NodePort
  ...
```

Please check cluster configuration for detailed configuration.

12.2.8.1.2 Stability Test Cases Added

- Stop all etcds and kubelets

12.2.8.1.3 Improvements

- Simplify GKE SSD setup
- Modularization for AWS Terraform scripts
- Turn on the automatic failover feature by default
- Enable configmap rollout by default
- Enable stable scheduling by default
- Support multiple TiDB clusters management in Alibaba Cloud
- Enable AWS NLB cross zone load balancing by default

12.2.8.1.4 Bug Fixes

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

12.2.8.2 Detailed Bug Fixes and Changes

- Allow upgrading TiDB monitor along with TiDB version (#666)
- Specify the TiKV status address to fix monitoring (#695)
- Fix sysbench installation on bastion machine for AWS deployment (#688)
- Update the `git add upstream` command to use `https` in contributing document (#690)
- Stability cases: stop kubelet and etcd (#665)
- Limit test cover packages (#687)
- Enable nlb cross zone load balancing by default (#686)
- Add TiKV raftstore parameters (#681)
- Support multiple TiDB clusters management for Alibaba Cloud (#658)
- Adjust the `EndEvictLeader` function (#680)
- Add more logs (#676)
- Update feature gates to support `key={true,false}` syntax (#677)
- Fix the typo meke to make (#679)
- Enable configmap rollout by default and quote configmap digest suffix (#678)
- Turn automatic failover on (#667)
- Sets node count for default pool equal to total desired node count (#673)
- Upgrade default TiDB version to v3.0.1 (#671)
- Remove storeLabels (#663)
- Change the way to configure TiDB/TiKV/PD in charts (#638)
- Modularize for AWS terraform scripts (#650)
- Change the `DeferClose` function (#653)
- Increase the default storage size for Pump from 10Gi to 20Gi in response to `stop-write-at-available-space` (#657)
- Simplify local SDD setup (#644)

12.2.9 TiDB Operator 1.0 RC.1 Release Notes

Release date: July 12, 2019

TiDB Operator version: 1.0.0-rc.1

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

12.2.9.1.1 Stability test cases added

- Stop kube-proxy
- Upgrade tidb-operator

12.2.9.1.2 Improvements

- Get the TS first and increase the TiKV GC life time to 3 hours before the full backup

- Add endpoints list and watch permission for controller-manager
- Scheduler image is updated to use "k8s.gcr.io/kube-scheduler" which is much smaller than "gcr.io/google-containers/hyperkube". You must pre-pull the new scheduler image into your airgap environment before upgrading.
- Full backup data can be uploaded to or downloaded from Amazon S3
- The terraform scripts support manage multiple TiDB clusters in one EKS cluster.
- Add `tikv.storeLables` setting
- On GKE one can use COS for TiKV nodes with small data for faster startup
- Support force upgrade when PD cluster is unavailable.

12.2.9.1.3 Bug Fixes

- Fix unbound variable in the backup script
- Give kube-scheduler permission to update/patch pod status
- Fix tidb user of scheduled backup script
- Fix scheduled backup to ceph object storage
- Fix several usability problems for AWS terraform deployment
- Fix scheduled backup bug: segmentation fault when backup user's password is empty

12.2.9.2 Detailed Bug Fixes and Changes

- Segmentation fault when backup user's password is empty (#649)
- Small fixes for terraform AWS (#646)
- TiKV upgrade bug fix (#626)
- Improve the readability of some code (#639)
- Support force upgrade when PD cluster is unavailable (#631)
- Add new terraform version requirement to AWS deployment (#636)
- GKE local ssd provisioner for COS (#612)
- Remove TiDB version from build (#627)
- Refactor so that using the PD API avoids unnecessary imports (#618)
- Add `storeLabels` setting (#527)
- Update google-kubernetes-tutorial.md (#622)
- Multiple clusters management in EKS (#616)
- Add Amazon S3 support to the backup/restore features (#606)
- Pass TiKV upgrade case (#619)
- Separate slow log with TiDB server log by default (#610)
- Fix the problem of unbound variable in backup script (#608)
- Fix notes of tidb-backup chart (#595)
- Give kube-scheduler ability to update/patch pod status. (#611)
- Use kube-scheduler image instead of hyperkube (#596)
- Fix pull request template grammar (#607)
- Local SSD provision: reduce network traffic (#601)
- Add operator upgrade case (#579)
- Fix a bug that TiKV status is always upgrade (#598)

- Build without debugger symbols (#592)
- Improve error messages (#591)
- Fix tidb user of scheduled backup script (#594)
- Fix dt case bug (#571)
- GKE terraform (#585)
- Fix scheduled backup to Ceph object storage (#576)
- Add stop kube-scheduler/kube-controller-manager test cases (#583)
- Add endpoints list and watch permission for controller-manager (#590)
- Refine fullbackup (#570)
- Make sure go modules files are always tidy and up to date (#588)
- Local SSD on GKE (#577)
- Stop kube-proxy case (#556)
- Fix resource unit (#573)
- Give local-volume-provisioner pod a QoS of Guaranteed (#569)
- Check PD endpoints status when it's unhealthy (#545)

12.2.10 TiDB Operator 1.0 Beta.3 Release Notes

Release date: June 6, 2019

TiDB Operator version: 1.0.0-beta.3

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

12.2.10.1.1 Action Required

- ACTION REQUIRED: `nodeSelectorRequired` was removed from values.yaml.
- ACTION REQUIRED: Comma-separated values support in `nodeSelector` has been dropped, please use new-added `affinity` field which has a more expressive syntax.

12.2.10.1.2 A lot of stability cases added

- ConfigMap rollout
- One PD replicas
- Stop TiDB Operator itself
- TiDB stable scheduling
- Disaster tolerance and data regions disaster tolerance
- Fix many bugs of stability test

12.2.10.1.3 New Features

- Introduce ConfigMap rollout management. With the feature gate open, configuration file changes will be automatically applied to the cluster via a rolling update. Currently, the `scheduler` and `replication` configurations of PD can not be changed via

ConfigMap rollout. You can use `pd-ctl` to change these values instead, see #487 for details.
- Support stable scheduling for pods of TiDB members in tidb-scheduler.
- Support adding additional pod annotations for PD/TiKV/TiDB, e.g. fluent-bit.io/parser.
- Support the affinity feature of k8s which can define the rule of assigning pods to nodes
- Allow pausing during TiDB upgrade

12.2.10.1.4 Documentation Improvement

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

12.2.10.1.5 Pass User Acceptance Tests

12.2.10.1.6 Other improvements

- Upgrade default TiDB version to v3.0.0-rc.1
- Fix a bug in reporting assigned nodes of TiDB members
- `tkctl get` can show cpu usage correctly now
- Adhoc backup now appends the start time to the PVC name by default.
- Add the privileged option for TiKV pod
- `tkctl upinfo` can show nodeIP podIP port now
- Get TS and use it before full backup using mydumper
- Fix capabilities issue for `tkctl debug` command

12.2.10.2 Detailed Bug Fixes and Changes

- Add capabilities and privilege mode for debug container (#537)
- Note helm versions in deployment docs (#553)
- Split public and private subnets when using existing vpc (#530)
- Release v1.0.0-beta.3 (#557)
- GKE terraform upgrade to 0.12 and fix bastion instance zone to be region agnostic (#554)
- Get TS and use it before full backup using mydumper (#534)
- Add port podip nodeip to tkctl upinfo (#538)
- Fix disaster tolerance of stability test (#543)
- Add privileged option for TiKV pod template (#550)
- Use staticcheck instead of megacheck (#548)
- Refine backup and restore documentation (#518)
- Fix stability tidb pause case (#542)

- Fix tkctl get cpu info rendering (#536)
- Fix Aliyun tf output rendering and refine documents (#511)
- Make webhook configurable (#529)
- Add pods disaster tolerance and data regions disaster tolerance test cases (#497)
- Remove helm hook annotation for initializer job (#526)
- Add stable scheduling e2e test case (#524)
- Upgrade TiDB version in related documentations (#532)
- Fix a bug in reporting assigned nodes of TiDB members (#531)
- Reduce wait time and fix stability test (#525)
- Fix documentation usability issues in GCP document (#519)
- PD replicas 1 and stop tidb-operator (#496)
- Pause-upgrade stability test (#521)
- Fix restore script bug (#510)
- Retry truncating sst files upon failure (#484)
- Upgrade default TiDB to v3.0.0-rc.1 (#520)
- Add `--namespace` when creating backup secret (#515)
- New stability test case for ConfigMap rollout (#499)
- Fix issues found in Queeny's test (#507)
- Pause rolling-upgrade process of TiDB statefulset (#470)
- GKE terraform and guide (#493)
- Support the affinity feature of Kubernetes which defines the rule of assigning pods to nodes (#475)
- Support adding additional pod annotations for PD/TiKV/TiDB (#500)
- Document PD configuration issue (#504)
- Refine Aliyun and AWS cloud TiDB configurations (#492)
- Update wording and add note (#502)
- Support stable scheduling for TiDB (#477)
- Fix `make lint` (#495)
- Support updating configuration on the fly (#479)
- Update AWS deploy docs after testing (#491)
- Add release-note to pull_request_template.md (#490)
- Design proposal of stable scheduling in TiDB (#466)
- Update DinD image to make it possible to configure HTTP proxies (#485)
- Fix a broken link (#489)
- Fix typo (#483)

12.2.11 TiDB Operator 1.0 Beta.2 Release Notes

Release date: May 10, 2019

TiDB Operator version: 1.0.0-beta.2

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

12.2.11.1.1 Stability has been greatly enhanced

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

12.2.11.1.2 Greatly improved ease of use

- One-command deployment for AWS, Aliyun
- Minikube deployment for testing
- Tkctl cli tool
- Refactor backup chart for ease use
- Refine initializer job
- Grafana monitor dashboard improved, support multi-version
- Improved user guide
- Contributing documentation

12.2.11.1.3 Bug fixes

- Fix PD start script, add join file when startup
- Fix TiKV failover take too long
- Fix PD ha when replcias is less than 3
- Fix a tidb-scheduler acquireLock bug and emit event when scheduled failed
- Fix scheduler ha bug with defer deleting pods
- Fix a bug when using shareinformer without deepcopy

12.2.11.1.4 Other improvements

- Remove pushgateway from TiKV pod
- Add GitHub templates for issue reporting and PR
- Automatically set the scheduler K8s version
- Switch to go module
- Support slow log of TiDB

12.2.11.2 Detailed Bug Fixes and Changes

- Don't initialize when there is no tidb.password (#282)
- Fix join script (#285)
- Document tool setup and e2e test detail in `CONTRIBUTING.md` (#288)
- Update setup.md (#281)
- Support slow log tailing sidcar for TiDB instance (#290)
- Flexible tidb initializer job with secret set outside of helm (#286)
- Ensure SLOW_LOG_FILE env variable is always set (#298)
- Fix setup document description (#300)
- Refactor backup (#301)

- Abandon vendor and refresh go.sum (#311)
- Set the SLOW_LOG_FILE in the startup script (#307)
- Automatically set the scheduler K8s version (#313)
- TiDB stability test main function (#306)
- Add fault-trigger server (#312)
- Add ad-hoc backup and restore function (#316)
- Add scale & upgrade case functions (#309)
- Add slack (#318)
- Log dump when test failed (#317)
- Add fault-trigger client (#326)
- Monitor checker (#320)
- Add blockWriter case for inserting data (#321)
- Add scheduled-backup test case (#322)
- Port ddl test as a workload (#328)
- Use fault-trigger at e2e tests and add some log (#330)
- Add binlog deploy and check process (#329)
- Fix e2e can not make (#331)
- Multi TiDB cluster testing (#334)
- Fix backup test bugs (#335)
- Delete `blockWrite.go` and use `blockwrite.go` instead (#333)
- Remove vendor (#344)
- Add more checks for scale & upgrade (#327)
- Support more fault injection (#345)
- Rewrite e2e (#346)
- Add failover test (#349)
- Fix HA when the number of replicas are less than 3 (#351)
- Add fault-trigger service file (#353)
- Fix dind doc (#352)
- Add additionalPrintColumns for TidbCluster CRD (#361)
- Refactor stability main function (#363)
- Enable admin privilege for prom (#360)
- Update `README.md` with new info (#365)
- Build CLI (#357)
- Add extraLabels variable in tidb-cluster chart (#373)
- Fix TiKV failover (#368)
- Separate and ensure setup before e2e-build (#375)
- Fix `codegen.sh` and lock related dependencies (#371)
- Add sst-file-corruption case (#382)
- Use release name as default clusterName (#354)
- Add util class to support adding annotations to Grafana (#378)
- Use Grafana provisioning to replace dashboard installer (#388)
- Ensure test env is ready before cases running (#386)
- Remove monitor config job check (#390)
- Update local-pv documentation (#383)
- Update Jenkins links in `README.md` (#395)

- Fix e2e workflow in `CONTRIBUTING.md` (#392)
- Support running stability test out of cluster (#397)
- Update TiDB secret docs and charts (#398)
- Enable blockWriter write pressure in stability test (#399)
- Support debug and ctop commands in CLI (#387)
- Marketplace update (#380)
- Update editable value from true to false (#394)
- Add fault inject for kube proxy (#384)
- Use `ioutil.TempDir()` create charts and operator repo's directories (#405)
- Improve workflow in docs/google-kubernetes-tutorial.md (#400)
- Support plugin start argument for TiDB instance (#412)
- Replace govet with official vet tool (#416)
- Allocate 24 PVs by default (after 2 clusters are scaled to (#407)
- Refine stability (#422)
- Record event as grafana annotation in stability test (#414)
- Add GitHub templates for issue reporting and PR (#420)
- Add TiDBUpgrading func (#423)
- Fix operator chart issue (#419)
- Fix stability issues (#433)
- Change cert generate method and add pd and kv prestop webhook (#406)
- A tidb-scheduler bug fix and emit event when scheduled failed (#427)
- Shell completion for tkctl (#431)
- Delete an duplicate import (#434)
- Add etcd and kube-apiserver faults (#367)
- Fix TiDB Slack link (#444)
- Fix scheduler ha bug (#443)
- Add terraform script to auto deploy TiDB cluster on AWS (#401)
- Add instructions to access Grafana in GKE tutorial (#448)
- Fix label selector (#437)
- No need to set ClusterIP when syncing headless service (#432)
- Document how to deploy TiDB cluster with tidb-operator in minikube (#451)
- Add slack notify (#439)
- Fix local dind env (#440)
- Add terraform scripts to support alibaba cloud ACK deployment (#436)
- Fix backup data compare logic (#454)
- Async emit annotations (#438)
- Use TiDB v2.1.8 by default & remove pushgateway (#435)
- Fix a bug that uses shareinformer without copy (#462)
- Add version command for tkctl (#456)
- Add tkctl user manual (#452)
- Fix binlog problem on large scale (#460)
- Copy kubernetes.io/hostname label to PVs (#464)
- AWS EKS tutorial change to new terraform script (#463)
- Update documentation of minikube installation (#471)
- Update documentation of DinD installation (#458)

- Add instructions to access Grafana (#476)
- Support-multi-version-dashboard (#473)
- Update Aliyun deploy docs after testing (#474)
- GKE local SSD size warning (#467)
- Update roadmap (#376)

12.2.12 TiDB Operator 1.0 Beta.1 P2 Release Notes

Release date: February 21, 2019

TiDB Operator version: 1.0.0-beta.1-p2

12.2.12.1 Notable Changes

- New algorithm for scheduler HA predicate (#260)
- Add TiDB discovery service (#262)
- Serial scheduling (#266)
- Change tolerations type to an array (#271)
- Start directly when where is join file (#275)
- Add code coverage icon (#272)
- In `values.yml`, omit just the empty leaves (#273)
- Charts: backup to ceph object storage (#280)
- Add `ClusterIDLabelKey` label to TidbCluster (#279)

12.2.13 TiDB Operator 1.0 Beta.1 P1 Release Notes

Release date: January 7, 2019

TiDB Operator version: 1.0.0-beta.1-p1

12.2.13.1 Bug Fixes

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

12.2.13.2 Docs

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

12.2.14 TiDB Operator 1.0 Beta.1 Release Notes

Release date: December 27, 2018

TiDB Operator version: 1.0.0-beta.1

12.2.14.1 Bug Fixes

- Fix pd_control bug: avoid relying on PD error response text (#197)
- Add orphan pod cleaner (#201)
- Fix scheduler configuration for Kubernetes 1.12 (#200)
- Fix Grafana configuration (#206)
- Fix pd failover bug: scale out directly when failover occurs (#217)
- Refactor PD failover (#211)
- Refactor tidb_cluster_control logic (#215)
- Fix upgrade logic: avoid updating pd/tikv/tidb simultaneously (#234)
- Fix PD control logic: get member/store before delete member/store and fix member id parse error (#245)
- Fix documents errors (#213)
- Fix backup and restore script bug (#251 #254 #255)
- Fix GKE multiple availability zones deployment PD disk scheduling bug (#248)

12.2.14.2 Minor Improvements

- Add Kubernetes 1.12 local DinD scripts (#195)
- Bump default TiDB to v2.1.0 (#212)
- Release tidb-operator/tidb-cluster charts (#216)
- Add connection timeout for TiDB password setter job (#219)
- Separate ad-hoc backup and restore to another chart (#227)
- Add compiler version info to tidb-operator binary (#237)
- Allow specifying TiDB service LoadBalancer IP (#246)
- Expose TiKV cpu/memory related configuration to values.yaml (#252)

12.2.15 TiDB Operator 1.0 Beta.0 Release Notes

Release date: November 26, 2018

TiDB Operator version: 1.0.0-beta.0

12.2.15.1 Notable Changes

- Introduce basic chaos testing
- Improve unit test coverage (#179 #181 #182 #184 #190 #192 #194)
- Add default value for log-level of PD/TiKV/TiDB (#185)
- Fix PD connection timeout issue for DinD environment (#186)
- Fix monitor configuration (#193)
- Fix document Helm client version requirement (#175)
- Keep scheduler name consistent in chart (#188)
- Remove unnecessary warning message when volumeName is empty (#177)
- Migrate to Go 1.11 module (#178)
- Add user guide (#187)

12.3 v0

12.3.1 TiDB Operator 0.4 Release Notes

Release date: November 9, 2018

TiDB Operator version: 0.4.0

12.3.1.1 Notable Changes

- Extend Kubernetes built-in scheduler for TiDB data awareness pod scheduling (#145)
- Restore backup data from GCS bucket (#160)
- Set password for TiDB when a TiDB cluster is first deployed (#171)

12.3.1.2 Minor Changes and Bug Fixes

- Update roadmap for the following two months (#166)
- Add more unit tests (#169)
- E2E test with multiple clusters (#162)
- E2E test for meta info synchronization (#164)
- Add TiDB failover limit (#163)
- Synchronize PV reclaim policy early to persist data (#169)
- Use helm release name as instance label (#168) (breaking change)
- Fix local PV setup script (#172)

12.3.2 TiDB Operator 0.3.1 Release Notes

Release date: October 31, 2018

TiDB Operator version: 0.3.1

12.3.2.1 Minor Changes

- Paramertize the serviceAccount (#116 #111)
- Bump TiDB to v2.0.7 & allow user specified config files (#121)
- Remove binding mode for GKE pd-ssd storageclass (#130)
- Modified placement of tidb_version (#125)
- Update google-kubernetes-tutorial.md (#105)
- Remove redundant creation statement of namespace tidb-operator-e2e (#132)
- Update the label name of app in local dind documentation (#136)
- Remove noisy events (#131)
- Marketplace (#123 #135)
- Change monitor/backup/binlog pvc labels (#143)
- TiDB readiness probes (#147)

- Add doc on how to provision kubernetes on AWS (#71)
- Add imagePullPolicy support (#152)
- Separation startup scripts and application config from yaml files (#149)
- Update marketplace for our open source offering (#151)
- Add validation to crd (#153)
- Marketplace: use the Release.Name (#157)

12.3.2.2 Bug Fixes

- Fix parallel upgrade bug (#118)
- Fix wrong parameter AGRS to ARGS (#114)
- Can't recover after a upgrade failed (#120)
- Scale in when store id match (#124)
- PD can't scale out if not all members are ready (#142)
- podLister and pvcLister usages are wrong (#158)

12.3.3 TiDB Operator 0.3.0 Release Notes

Release date: October 12, 2018

TiDB Operator version: 0.3.0

12.3.3.1 Notable Changes

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

12.3.4 TiDB Operator 0.2.1 Release Notes

Release date: September 20, 2018

TiDB Operator version: 0.2.1

12.3.4.1 Bug Fixes

- Fix retry on conflict logic (#87)
- Fix TiDB timezone configuration by setting TZ environment variable (#96)
- Fix failover by keeping spec replicas unchanged (#95)
- Fix repeated updating pod and pd/tidb StatefulSet (#101)

12.3.5 TiDB Operator 0.2.0 Release Notes

Release date: September 11, 2018

TiDB Operator version: 0.2.0

12.3.5.1 Notable Changes

- Support auto-failover experimentally
- Unify Tiller managed resources and TiDB Operator managed resources labels (breaking change)
- Manage TiDB service via Tiller instead of TiDB Operator, allow more parameters to be customized (required for public cloud load balancer)
- Add toleration for TiDB cluster components (useful for dedicated deployment)
- Add script to easy setup DinD environment
- Lint and format code in CI
- Refactor upgrade functions as interface

12.3.6 TiDB Operator 0.1.0 Release Notes

Release date: August 22, 2018

TiDB Operator version: 0.1.0

12.3.6.1 Notable Changes

- Bootstrap multiple TiDB clusters
- Monitor deployment support
- Helm charts support
- Basic Network PV/Local PV support
- Safely scale the TiDB cluster
- Upgrade the TiDB cluster in order
- Stop the TiDB process without terminating Pod
- Synchronize cluster meta info to POD/PV/PVC labels
- Basic unit tests & E2E tests
- Tutorials for GKE, local DinD
