TiDB on Kubernetes Documentation

PingCAP Inc.

20250123

Table of Contents

1 TiDB on Kubernetes Docs

2 Introduction

2.1 TiDB Operator Overview

TiDB Operator is an automatic operation system for TiDB clusters on 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 cloud or in a self-managed environment.

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

TiDB versions	Compatible TiDB Operator versions
dev	dev
TiDB >= 8.0	1.6 (Recommended), 1.5
7.1 <= TiDB < 8.0	1.5 (Recommended), 1.4
6.5 <= TiDB < 7.1	1.5, 1.4 (Recommended), 1.3
5.4 <= TiDB < 6.5	1.4, 1.3 (Recommended)

TiDB versions	Compatible TiDB Operator versions
5.1 <= TiDB < 5.4	1.4, 1.3 (Recommended), 1.2 (End of support)
3.0 <= TiDB < 5.1	1.4, 1.3 (Recommended), 1.2 (End of support), 1.1 (End of support)
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 on 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 Google Cloud GKE
    * Deploy TiDB on Azure AKS
  - In an existing Kubernetes cluster:

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

    You also need to adjust the configuration of the Kubernetes cluster based on Prerequisites for TiDB on 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 on Kubernetes:

- Access the TiDB Cluster
- Scale TiDB Cluster
- Upgrade a TiDB Cluster
- Change the Configuration of TiDB Cluster
- Back up and Restore a TiDB Cluster
- Automatic Failover
- Monitor a TiDB Cluster on Kubernetes
- View TiDB Logs on Kubernetes
- Maintain Kubernetes Nodes that Hold the TiDB Cluster

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

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

Some of TiDB's tools are used differently on Kubernetes. You can see Tools on Kubernetes to understand how TiDB tools are used on 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 v1.6

TiDB Operator 1.6 introduces the following key features, which helps you manage TiDB clusters and the tools more easily in terms of extensibility and usability.

2.2.1 Compatibility changes

- Upgrade Kubernetes dependency to v1.28, and it is not recommended to deploy `tidb-scheduler`.
- When deploying using Helm chart, support setting lock resource used by `tidb-controller-manager` for leader election, with the default value of `.Values.controllerManager.leaderResourceLock: leases`. When upgrading TiDB Operator to v1.6.0-beta.1 or a later version, it is recommended to first set `.Values.controllerManager.leaderResourceLock: endpointsleases` and wait for the new `tidb-controller-manager` to run normally before setting it to `.Values.controllerManager.leaderResourceLock: leases` to update the deployment.

2.2.2 Extensibility

- Support deploying PD v8.0.0 and later versions in microservice mode (experimental).
- Support scaling out or in TiDB components in parallel.

2.2.3 Usability

- Support automatically setting location labels for TiProxy.
- Support setting `maxSkew`, `minDomains`, and `nodeAffinityPolicy` in `topologySpreadConstraints` for components of a TiDB cluster.
- Support setting `startupProbe` for TiDB components.
- Support setting additional command-line arguments for TiDB components.
- Support setting `livenessProbe` and `readinessProbe` for the Discovery component.
- Support setting `nodeSelector` for the TidbInitializer component.
- Enable TiFlash to directly mount ConfigMap without relying on an InitContainer to process configuration files.

3 Get Started with TiDB on Kubernetes

This document introduces how to create a simple Kubernetes cluster and use it to deploy a basic test TiDB cluster using TiDB Operator.

Warning:

> This document is for demonstration purposes only. **Do not** follow it in production environments. For deployment in production environments, refer to the instructions in See also.

To deploy TiDB Operator and a TiDB cluster, follow these steps:

1. Create a test Kubernetes 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 the TiDB cluster and the Kubernetes cluster

You can watch the following video (approximately 12 minutes) to learn how to get started with TiDB Operator.

3.1 Step 1: Create a test Kubernetes cluster

This section describes two methods for creating a simple Kubernetes cluster. After creating a Kubernetes cluster, you can use it to test TiDB clusters managed by TiDB Operator. Choose the method that best suits your environment.

- Method 1: Create a Kubernetes cluster using kind: Deploy a Kubernetes cluster in Docker using kind, a common and recommended method.
- Method 2: Create a Kubernetes cluster using minikube: Deploy a Kubernetes cluster locally in a VM using minikube.

Alternatively, you can deploy a Kubernetes cluster on Google Kubernetes Engine on Google Cloud using the Google Cloud Shell.

3.1.1 Method 1: Create a Kubernetes cluster using kind

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

kind is a popular tool for running local Kubernetes clusters using Docker containers as cluster nodes. For available tags, see Docker Hub. The latest version of kind is used by default.

Before deployment, ensure that the following requirements are met:

- Docker: version >= 18.09
- kubectl: version >= 1.24

- kind: version >= 0.19.0
- For Linux, the value of the sysctl parameter net.ipv4.ip_forward should be set to `1`.

Here is an example using `kind` v0.19.0:

```
kind create cluster
```

Expected output

```
Creating cluster "kind" ...
   Ensuring node image (kindest/node:v1.27.1)
   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 are now ready to deploy TiDB Operator.

3.1.2 Method 2: Create a Kubernetes cluster using minikube

You can create a Kubernetes cluster in a VM using minikube, which supports macOS, Linux, and Windows.

Before deployment, ensure that the following requirements are met:

- minikube: version 1.0.0 or later versions. Newer versions like v1.24 are recommended. minikube requires a compatible hypervisor. For details, refer to minikube installation instructions.
- kubectl: version >= 1.24

3.1.2.1 Start a minikube Kubernetes cluster

After installing minikube, run the following command to start a minikube Kubernetes cluster:

```
minikube start
```

3.1.2.2 Use `kubectl` to interact with the 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 run the following alias definition command after opening a new shell.

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

Run the following command to check the status of Kubernetes and ensure that `kubectl` can connect to it:

```
kubectl cluster-info
```

Expected 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.

3.2 Step 2: Deploy TiDB Operator

To deploy TiDB Operator, you need to follow these steps:

3.2.1 Install TiDB Operator CRDs

First, you need to install the Custom Resource Definitions (CRDs) that are required for TiDB Operator. These CRDs implement different components of the TiDB cluster.

To install the CRDs, run the following command:

```
kubectl create -f https://raw.githubusercontent.com/pingcap/tidb-operator/v1
    ↪ .6.1/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

To install TiDB Operator, you can use Helm 3. Follow these steps:

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 v1.6.1
```

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, run 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
```

Once all the Pods are in the "Running" state, you can proceed to the next step.

3.3 Step 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/v1.6.1/examples/basic/tidb-cluster.yaml
```

Expected output

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

If you need to deploy a TiDB cluster on an ARM64 machine, refer to Deploying a TiDB Cluster on ARM64 Machines.

Note:

Starting from v8.0.0, PD supports the microservice mode (experimental). To deploy PD microservices, use the following command:

```
kubectl create namespace tidb-cluster && \
    kubectl -n tidb-cluster apply -f https://raw.
        ↪ githubusercontent.com/pingcap/tidb-operator/v1.6.1/
        ↪ examples/basic/pd-micro-service-cluster.yaml
```

3.3.2 Deploy TiDB Dashboard independently

```
kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com/pingcap/
    ↪ tidb-operator/v1.6.1/examples/basic/tidb-dashboard.yaml
```

Expected output

```
tidbdashboard.pingcap.com/basic created
```

3.3.3 Deploy TiDB monitoring services

```
kubectl -n tidb-cluster apply -f https://raw.githubusercontent.com/pingcap/
    ↪ tidb-operator/v1.6.1/examples/basic/tidb-monitor.yaml
```

Expected output

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

3.3.4 View the Pod status

```
watch kubectl get po -n 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
```

Wait until all Pods for each service are started. Once you see that the Pods for each type (`-pd`, `-tikv`, and `-tidb`) are in the "Running" state, you can press Ctrl+C to return to the command line and proceed with connecting to your TiDB cluster.

3.4 Step 4: Connect to TiDB

To connect to TiDB, you can use the MySQL client since TiDB supports the MySQL protocol and most of its syntax.

3.4.1 Install the MySQL client

Before connecting to TiDB, make sure you have 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.

3.4.2 Forward port 4000

To connect to TiDB, you need to forward a port from the local host to the TiDB service on 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 14000:4000 > pf14000.out
    ↪  &
```

If port `14000` is already occupied, you can replace it with an available port. This command runs in the background and writes its output to a file named `pf14000.out`. You can continue to run the command in the current shell session.

3.4.3 Connect to the TiDB service

> **Note:**
>
> To connect to TiDB (version < 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 --comments -h 127.0.0.1 -P 14000 -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-v4.0.0 MySQL Community Server (Apache License
    ↪ 2.0)

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 run the following commands to verify that some features are available in TiDB. Note that some commands require TiDB 4.0 or higher versions. If you have deployed an earlier version, you need to upgrade the TiDB cluster.

Create ahello_worldtable

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

mysql> select * from information_schema.tikv_region_status where db_name=
    ↪ database() and table_name='hello_world'\G
*************************** 1. row ***************************
       REGION_ID: 2
       START_KEY: 7480000000000000FF3700000000000000F8
         END_KEY:
        TABLE_ID: 55
         DB_NAME: test
      TABLE_NAME: hello_world
        IS_INDEX: 0
        INDEX_ID: NULL
      INDEX_NAME: NULL
  EPOCH_CONF_VER: 5
   EPOCH_VERSION: 23
   WRITTEN_BYTES: 0
      READ_BYTES: 0
APPROXIMATE_SIZE: 1
APPROXIMATE_KEYS: 0
1 row in set (0.03 sec)
```

Query the TiDB version

```
mysql> select tidb_version()\G
*************************** 1. row ***************************
        tidb_version(): Release Version: v8.5.0
               Edition: Community
       Git Commit Hash: d13e52ed6e22cc5789bed7c64c861578cd2ed55b
            Git Branch: heads/refs/tags/v8.5.0
        UTC Build Time: 2024-12-19 14:38:24
             GoVersion: go1.23.2
          Race Enabled: false
Check Table Before Drop: false
                 Store: tikv
1 row in set (0.01 sec)
```

Query the TiKV store status

```
mysql> select * from information_schema.tikv_store_status\G
*************************** 1. row ***************************
           STORE_ID: 4
            ADDRESS: basic-tikv-0.basic-tikv-peer.tidb-cluster.svc:20160
        STORE_STATE: 0
   STORE_STATE_NAME: Up
              LABEL: null
            VERSION: 5.2.1
           CAPACITY: 58.42GiB
          AVAILABLE: 36.18GiB
       LEADER_COUNT: 3
      LEADER_WEIGHT: 1
       LEADER_SCORE: 3
        LEADER_SIZE: 3
       REGION_COUNT: 21
      REGION_WEIGHT: 1
       REGION_SCORE: 21
        REGION_SIZE: 21
           START_TS: 2020-05-28 22:48:21
  LAST_HEARTBEAT_TS: 2020-05-28 22:52:01
             UPTIME: 3m40.598302151s
1 rows in set (0.01 sec)
```

Query the TiDB cluster information

This command is effective only in TiDB 4.0 or later versions. If your TiDB does not support the command, you need to 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: 5.2.1
       GIT_HASH: 689a6b6439ae7835947fcaccf329a3fc303986cb
     START_TIME: 2020-05-28T22:50:11Z
         UPTIME: 3m21.459090928s
*************************** 2. row ***************************
           TYPE: pd
       INSTANCE: basic-pd:2379
  STATUS_ADDRESS: basic-pd:2379
        VERSION: 5.2.1
       GIT_HASH: 56d4c3d2237f5bf6fb11a794731ed1d95c8020c2
     START_TIME: 2020-05-28T22:45:04Z
         UPTIME: 8m28.459091915s
```

```
*************************** 3. row ***************************
           TYPE: tikv
       INSTANCE: basic-tikv-0.basic-tikv-peer.tidb-cluster.svc:20160
  STATUS_ADDRESS: 0.0.0.0:20180
        VERSION: 5.2.1
       GIT_HASH: 198a2cea01734ce8f46d55a29708f123f9133944
     START_TIME: 2020-05-28T22:48:21Z
         UPTIME: 5m11.459102648s
3 rows in set (0.01 sec)
```

3.4.4 Access the Grafana dashboard

To access the Grafana dashboard locally, you need to forward the port for Grafana:

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

You can access the Grafana dashboard at http://localhost:3000 on the host where you run `kubectl`. The default username and password in Grafana are both `admin`.

Note that if you run `kubectl` in a Docker container or on a remote host instead of your local host, you cannot access the Grafana dashboard at http://localhost:3000 from your browser. In this case, you can run the following command to listen on all addresses:

```
kubectl port-forward --address 0.0.0.0 -n tidb-cluster svc/basic-grafana
    ↪ 3000 > pf3000.out &
```

Then access Grafana through http://$%7Bremote-server-IP%7D:3000.

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

3.4.5 Access the TiDB Dashboard web UI

To access the TiDB Dashboard web UI locally, you need to forward the port for TiDB Dashboard:

```
kubectl port-forward -n tidb-cluster svc/basic-tidb-dashboard-exposed 12333
    ↪ > pf12333.out &
```

You can access the panel of TiDB Dashboard at http://localhost:12333 on the host where you run `kubectl`.

Note that if you run `kubectl port-forward` in a Docker container or on a remote host instead of your local host, you cannot access TiDB Dashboard using `localhost` from your local browser. In this case, you can run the following command to listen on all addresses:

```
kubectl port-forward --address 0.0.0.0 -n tidb-cluster svc/basic-tidb-
    ↪ dashboard-exposed 12333 > pf12333.out &
```

Then access TiDB Dashboard through `http://${remote-server-IP}:12333`.

3.5 Step 5: Upgrade a TiDB cluster

TiDB Operator simplifies the process of performing a rolling upgrade of a TiDB cluster. This section describes how to upgrade your TiDB cluster to the "nightly" release.

Before proceeding, it is important to familiarize yourself with the `kubectl patch` subcommand. This command lets you directly apply changes to the running cluster resources. There are different patch strategies available, each with its own capabilities, limitations, and allowed formats. For more information, refer to the Kubernetes Patch document.

3.5.1 Modify the TiDB cluster version

To update the version of the TiDB cluster to "nightly," you can use a JSON merge patch. Execute the following command:

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

Expected output

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

3.5.2 Wait for Pods to restart

To monitor the progress of the cluster upgrade and observe the restart of its components, run the following command. You should see some Pods transitioning from `Terminating` to `ContainerCreating` and finally to `Running`.

```
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

Once all Pods have been restarted, you can verify that the cluster's version number has been updated.

Note that if you had previously set up port forwarding, you will need to reset it because the Pods it forwarded to have been destroyed and recreated.

```
kubectl port-forward -n tidb-cluster svc/basic-tidb 24000:4000 > pf24000.out
    ↪ &
```

If port 24000 is already in use, you can replace it with an available port.

3.5.4 Check the TiDB cluster version

To confirm the TiDB cluster's version, execute the following command:

```
mysql --comments -h 127.0.0.1 -P 24000 -u root -e 'select tidb_version()\G'
```

Expected output

Note that `nightly` is not a fixed version and the version might vary depending on the time the command is run.

```
*************************** 1. row ***************************
tidb_version(): Release Version: v8.5.0
Edition: Community
Git Commit Hash: d13e52ed6e22cc5789bed7c64c861578cd2ed55b
Git Branch: heads/refs/tags/v8.5.0
UTC Build Time: 2024-12-19 14:38:24
GoVersion: go1.23.2
Race Enabled: false
Check Table Before Drop: false
Store: tikv
```

3.6 Step 6: Destroy the TiDB cluster and the Kubernetes cluster

After you finish testing, you can destroy the TiDB cluster and the Kubernetes cluster.

3.6.1 Destroy the TiDB cluster

To destroy the TiDB cluster, follow these steps:

3.6.1.1 Stop `kubectl` port forwarding

If you have any running `kubectl` processes that are forwarding ports, make sure to end them by running the following command:

```
pgrep -lfa kubectl
```

3.6.1.2 Delete the TiDB cluster

To delete the TiDB cluster, use the following command:

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

In this command, `tc` is short for `tidbclusters`.

3.6.1.3 Delete TiDB monitoring services

To delete the TiDB monitoring services, run the following command:

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

3.6.1.4 Delete PV data

If your deployment includes persistent data storage, deleting the TiDB cluster does not remove the data in the cluster. If you do not need the data, you can clean it by running the following commands:

```
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"}}'
```

3.6.1.5 Delete namespaces

To ensure that there are no remaining resources, delete the namespace used for your TiDB cluster by running the following command:

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

3.6.2 Destroy the Kubernetes cluster

The method for destroying a Kubernetes cluster depends on how it was created. Here are the steps for destroying a Kubernetes cluster based on the creation method:

If you created the Kubernetes cluster using kind, use the following command to destroy it:

```
kind delete cluster
```

If you created the Kubernetes cluster using minikube, use the following command to destroy it:

```
minikube delete
```

3.7 See also

If you are interested in deploying a TiDB cluster in production environments, refer to the following documents:

On public clouds:

- Deploy TiDB on AWS EKS
- Deploy TiDB on Google Cloud GKE
- Deploy TiDB on Azure AKS

In a self-managed Kubernetes cluster:

- Familiarize yourself with the Prerequisites for TiDB on Kubernetes
- Configure the local PV for your Kubernetes cluster to achieve high performance for TiKV
- Deploy TiDB Operator on Kubernetes
- Deploy TiDB on General Kubernetes

4 Deploy

4.1 On Self-Managed Kubernetes

4.1.1 Prerequisites for TiDB on Kubernetes

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

4.1.1.1 Software version

Software Name	Version
Kubernetes	v1.24+
CentOS	7.6 and kernel 3.10.0-957 or later
Helm	v3.0.0+

4.1.1.2 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
```

4.1.1.3 Configure Iptables

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

```
iptables -P FORWARD ACCEPT
```

4.1.1.4 Disable SELinux

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

4.1.1.5 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
```

4.1.1.6 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
```

4.1.1.7 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
```

4.1.1.8 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
```

4.1.1.9 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
```

4.1.1.10 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
```

4.1.1.11 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.

4.1.1.12 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.

4.1.2 Persistent Storage Class Configuration on Kubernetes

TiDB cluster components such as PD, TiKV, TiDB monitoring, TiDB Binlog, and `tidb-backup` require persistent storage for data. To achieve this on Kubernetes, you need to use PersistentVolume (PV). Kubernetes supports different types of storage classes, which can be categorized into two main types:

- Network storage

  Network storage is not located on the current node but is mounted to the node through the network. It usually has redundant replicas to ensure high availability. In the event of a node failure, the corresponding network storage can be remounted to another node for continued use.

- Local storage

  Local storage is located on the current node and typically provides lower latency compared to network storage. However, it does not have redundant replicas, so data might be lost if the node fails. If the node is an IDC server, data can be partially restored, but if it is a virtual machine using local disk on a public cloud, data cannot be retrieved after a node failure.

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

> **Warning:**
>
> Do not delete a PV under any circumstances unless you are familiar with the underlying volume provisioner. Manually deleting a PV can result in orphaned volumes and unexpected behavior.

4.1.2.1 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 it is strongly recommended to use local SSD storage in a production environment.

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

When deploying TiDB clusters or `tidb-backup` with TiDB Operator, you can configure the `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.

4.1.2.2 Network PV configuration

To enable volume expansion for the corresponding `StorageClass`, run the following command:

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

After enabling volume expansion, you can expand the PV using the following method:

1. Edit the PersistentVolumeClaim (PVC) object:

   Suppose the PVC is currently 10 Gi and you 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}` still shows the original size. However, if you run the following command to view the size of the PV, it shows that the size has been expanded to the expected value.

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

4.1.2.3 Local PV configuration

Currently, Kubernetes supports statically allocated local storage. To create a local storage object, use `local-volume-provisioner` in the local-static-provisioner repository.

4.1.2.3.1 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 recommended to allocate a dedicated disk for TiDB, with SSD being the recommended disk type.

- For a disk that stores PD data, follow the steps to mount the disk. First, create multiple directories on 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. Each directory has a corresponding PV created, and each PD server uses one PV.

- For a disk that stores monitoring data, follow the steps to mount the disk. First, create multiple directories on 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. Each directory has a corresponding PV created, and each TiDB cluster's monitoring data uses one PV.

- For a disk that stores TiDB Binlog and backup data, follow the steps to mount the disk. First, create multiple directories on the disk and bind mount the directories 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. Each directory has a corresponding PV created, and each Pump and Drainer use one PV. All Ad-hoc full backup tasks and 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. For each subdirectory in the discovery directory, local-volume-provisioner creates a corresponding PV.

4.1.2.3.2 Step 2: Deploy local-volume-provisioner

Online deployment

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

```
wget https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/local-pv/local-volume-provisioner.yaml
```

2. If you are using the same discovery directory as described in Step 1: Pre-allocate local storage, you can skip this step. If you are using a different path for the 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 options for the local-volume-provisioner, refer to the Configuration document.

   - Modify the `volumes` and `volumeMounts` fields in the DaemonSet spec to ensure that 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 the `local-volume-provisioner`.

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.6.1/manifests/local-dind/local-volume-provisioner.
    ↪ yaml
```

4. Check the status of the 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
```

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

Note:

If there are no mount points in the discovery directory, no PV is created and the output is empty.

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

Offline deployment

The steps for offline deployment are the same as for online deployment, except for the following:

- Download the `local-volume-provisioner.yaml` file on a machine with Internet access, then upload it to the server and install it.
- The `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:v2.5.0` 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:v2.5.0
docker save -o local-volume-provisioner-v2.5.0.tar quay.io/external_storage/local-volume-provisioner:v2.5.0
```

  Copy the `local-volume-provisioner-v2.5.0.tar` file to the server, and execute the `docker load` command to load the file on the server:

```
docker load -i local-volume-provisioner-v2.5.0.tar
```

4.1.2.3.3 Best practices

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

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

4.1.2.4 Data safety

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

- To configure globally:

  The reclaim policy of a `StorageClass` is set at creation time and cannot be updated once created. If it is not set during creation, you can create another `StorageClass` with 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 a reclaim policy of `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
```

- To 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`.

4.1.2.4.1 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 the PV and its corresponding data by following these 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`. This automatically deletes and reclaims the PV.

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

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

4.1.3 Deploy TiDB Operator on Kubernetes

This document describes how to deploy TiDB Operator on Kubernetes.

4.1.3.1 Prerequisites

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

- Kubernetes >= v1.24
- DNS addons
- PersistentVolume
- RBAC enabled (optional)
- Helm 3

4.1.3.1.1 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 v1.24 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 Google Cloud GKE

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.

It is recommended to enable RBAC in the Kubernetes cluster.

4.1.3.1.2 Install Helm

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

4.1.3.2 Deploy TiDB Operator

4.1.3.2.1 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 create -f https://raw.githubusercontent.com/pingcap/tidb-operator/v1
    ↪ .6.1/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/v1.6.1/
    ↪ manifests/crd.yaml
kubectl create -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
```

4.1.3.2.2 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.

When you use TiDB Operator, `tidb-scheduler` is not mandatory. Refer to tidb-scheduler and default-scheduler to confirm whether you need to deploy `tidb-scheduler`. If you do not need `tidb-scheduler`, you can configure `scheduler.create: false` in the `values.yaml` file, so `tidb-scheduler` is not deployed.

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, `v1.6.1`. You can view the currently supported versions by running the `helm search repo -l tidb-operator` command.

2. Configure TiDB Operator

   TiDB Operator manages all TiDB clusters in the Kubernetes cluster by default. If you only need it to manage clusters in a specific namespace, you can set `clusterScoped: false` in `values.yaml`.

> **Note:**
>
> After setting `clusterScoped: false`, TiDB Operator will still operate Nodes, Persistent Volumes, and Storage Classes in the Kubernetes cluster by default. If the role that deploys TiDB Operator does not have the permissions to operate these resources, you can set the corresponding permission request under `controllerManager.clusterPermissions` to `false` to disable TiDB Operator's operations on these resources.

   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-v1.6.1.tgz
   ```

   Copy the `tidb-operator-v1.6.1.tgz` file to the target server and extract it to the current directory:

   ```
   tar zxvf tidb-operator.v1.6.1.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:v1.6.1
   pingcap/tidb-backup-manager:v1.6.1
   bitnami/kubectl:latest
   pingcap/advanced-statefulset:v0.7.0
   ```

   Next, download all these images using the following command:

   ```
   docker pull pingcap/tidb-operator:v1.6.1
   docker pull pingcap/tidb-backup-manager:v1.6.1
   docker pull bitnami/kubectl:latest
   docker pull pingcap/advanced-statefulset:v0.7.0

   docker save -o tidb-operator-v1.6.1.tar pingcap/tidb-operator:v1.6.1
   docker save -o tidb-backup-manager-v1.6.1.tar pingcap/tidb-backup-
       ↪ manager:v1.6.1
   docker save -o bitnami-kubectl.tar bitnami/kubectl:latest
   docker save -o advanced-statefulset-v0.3.3.tar pingcap/advanced-
       ↪ statefulset:v0.7.0
   ```

   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-v1.6.1.tar
docker load -i tidb-backup-manager-v1.6.1.tar
docker load -i bitnami-kubectl.tar
docker load -i advanced-statefulset-v0.3.3.tar
```

3. Configure TiDB Operator

   Modify the `./tidb-operator/values.yaml` file to configure TiDB Operator.

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
```

4.1.3.3 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
```

4.1.4 Configure a TiDB Cluster on 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

4.1.4.1 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 on 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.

4.1.4.2 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.

4.1.4.2.1 Cluster name

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

4.1.4.2.2 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 `v8.5.0`
- `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 `v8.5.0`

4.1.4.2.3 Recommended configuration

configUpdateStrategy

The default value of the `spec.configUpdateStrategy` field is `InPlace`, which means that when you modify `config` of a component, you need to manually trigger a rolling update to apply the new configurations to the cluster.

It is recommended that you configure `spec.configUpdateStrategy: RollingUpdate` to enable automatic update of configurations. In this way, every time the `config` of a component is updated, TiDB Operator automatically triggers a rolling update for the component and applies the modified configuration 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

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`.

startScriptVersion

To choose the different versions of the startup scripts for each component, you can configure the `spec.startScriptVersion` field in the cluster spec.

The supported versions of the start script are as follows:

- `v1` (default): the original version of the startup script.
- `v2`: to optimize the start script for each component and make sure that upgrading TiDB Operator does not result in cluster rolling restart, TiDB Operator v1.4.0 introduces `v2`. Compared to `v1`, `v2` has the following optimizations:
  - Use `dig` instead of `nslookup` to resolve DNS.
  - All components support debug mode.

It is recommended that you configure `spec.startScriptVersion` as the latest version (`v2`) for the new cluster.

> **Warning:**
>
> Modify the `startScriptVersion` field of the deployed cluster will cause the rolling restart.

4.1.4.2.4 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 on 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/ticdc.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:

To mount multiple PVs for TiKV:

```
tikv:
  ...
  config: |
    [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"
```

To mount multiple PVs for TiDB:

```
tidb:
  config: |
    path = "/tidb/data"
    [log.file]
      filename = "/tidb/log/tidb.log"
  storageVolumes:
  - name: data
    storageSize: "2Gi"
    mountPath: "/tidb/data"
  - name: log
    storageSize: "2Gi"
    mountPath: "/tidb/log"
```

To mount multiple PVs for PD:

```
pd:
  config: |
    data-dir = "/pd/data"
    [log.file]
      filename = "/pd/log/pd.log"
  storageVolumes:
  - name: data
    storageSize: "10Gi"
    mountPath: "/pd/data"
  - name: log
    storageSize: "10Gi"
    mountPath: "/pd/log"
```

To mount multiple PVs for TiCDC:

```
ticdc:
  ...
  config:
    dataDir: /ticdc/data
    logFile: /ticdc/log/cdc.log
  storageVolumes:
  - name: data
    storageSize: "10Gi"
    storageClassName: local-storage
    mountPath: "/ticdc/data"
  - name: log
    storageSize: "10Gi"
    storageClassName: local-storage
    mountPath: "/ticdc/log"
```

To mount multiple PVs for PD microservices (taking the `tso` microservice as an example):

> **Note:**
>
> Starting from v8.0.0, PD supports the microservice mode (experimental).

```
pd:
  mode: "ms"
pdms:
- name: "tso"
  config: |
    [log.file]
      filename = "/pdms/log/tso.log"
  storageVolumes:
  - name: log
    storageSize: "10Gi"
    mountPath: "/pdms/log"
```

> **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`.

4.1.4.2.5 HostNetwork

For PD, TiKV, TiDB, TiFlash, TiProxy, 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.

4.1.4.2.6 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:

```
spec:
  discovery:
    limits:
      cpu: "0.2"
    requests:
      cpu: "0.2"
  ...
```

4.1.4.2.7 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 PD microservices

> **Note:**
>
> Starting from v8.0.0, PD supports the microservice mode (experimental).

To enable PD microservices in your cluster, configure `spec.pd.mode` and `spec.pdms` in the `${cluster_name}/tidb-cluster.yaml` file:

```
spec:
  pd:
    mode: "ms"
  pdms:
```

```
  - name: "tso"
    baseImage: pingcap/pd
    replicas: 2
  - name: "scheduling"
    baseImage: pingcap/pd
    replicas: 1
```

- `spec.pd.mode` is used to enable or disable PD microservices. Setting it to `"ms"` enables PD microservices, while setting it to `""` or removing this field disables PD microservices.
- `spec.pdms.config` is used to configure PD microservices, and the specific configuration parameters are the same as `spec.pd.config`. To get all the parameters that can be configured for PD microservices, see the PD configuration file.

Enable TiProxy

The deployment method is the same as that of PD. In addition, you need to modify `spec.tiproxy` to manually specify the number of TiProxy components.

```
  tiproxy:
    baseImage: pingcap/tiproxy
    replicas: 3
    config:
```

When deploying TiProxy, you also need to configure additional parameters for TiDB. For detailed configuration steps, refer to Deploy TiProxy Load Balancer for an Existing TiDB Cluster.

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: 0
    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: 0
  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
```

4.1.4.2.8 Configure TiDB components

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

Configure TiDB parameters

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

For example:

```
spec:
  tidb:
    config: |
      split-table = true
      oom-action = "log"
```

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:

```
spec:
  tikv:
    config: |
      [storage]
        [storage.block-cache]
          capacity = "16GB"
```

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:

```
spec:
  pd:
    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 PD microservices

Note:

Starting from v8.0.0, PD supports the microservice mode (experimental).

You can configure PD microservice using the `spec.pd.mode` and `spec.pdms` parameters of the TidbCluster CR. Currently, PD supports two microservices: the `tso` microservice and the `scheduling` microservice. The configuration example is as follows:

```
spec:
  pd:
    mode: "ms"
  pdms:
  - name: "tso"
    baseImage: pingcap/pd
    replicas: 2
    config: |
      [log.file]
        filename = "/pdms/log/tso.log"
  - name: "scheduling"
    baseImage: pingcap/pd
    replicas: 1
    config: |
      [log.file]
        filename = "/pdms/log/scheduling.log"
```

In the preceding configuration, `spec.pdms` is used to configure PD microservices, and the specific configuration parameters are the same as `spec.pd.config`. To get all the parameters that can be configured for PD microservices, see the 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 microservice components might fail to start. This step is meant to be compatible with `Helm` deployment.
- If you enable the PD microservice mode when you deploy a TiDB cluster, some configuration items of PD microservices are persisted in etcd. The persisted configuration in etcd takes precedence over that in PD.
- If you enable the PD microservice mode for an existing TiDB cluster, some configuration items of PD microservices adopt the same values in PD configuration and are persisted in etcd. The persisted configuration in etcd takes precedence over that in PD.
- Hence, after the first startup of PD microservices, you cannot modify these configuration items using parameters. Instead, you can modify them dynamically using SQL statements, pd-ctl, or PD server API. Currently, among all the configuration items listed in Modify PD configuration dynamically, except `log.level`, all the other configuration items

cannot be modified using parameters after the first startup of PD microservices.

Configure TiProxy parameters

TiProxy parameters can be configured by `spec.tiproxy.config` in TidbCluster Custom Resource.

For example:

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

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

Configure TiFlash parameters

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

For example:

```
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 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:

For TiDB Operator v1.2.0-rc.2 and later versions, configure the parameters in the TOML format as follows:

```
spec:
  ticdc:
    config: |
      gc-ttl = 86400
      log-level = "info"
```

For TiDB Operator versions earlier than v1.2.0-rc.2, configure the parameters in the YAML format as follows:

```
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
```

Note:

For the following cases, configure `maxFailoverCount: 0` explicitly:

- The Kubernetes cluster does not have enough resources for TiDB Operator to scale out the new Pod. In such cases, the new Pod will be in the Pending state.
- You do not want to enable the automatic failover function.

4.1.4.2.9 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.

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.

```
spec:
  tidb:
    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.

4.1.4.2.10 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 (1500 minutes by default), TiKV Pod is restarted. If TiKV has fewer than 2 replicas, TiDB Operator forces an upgrade without waiting for the timeout.

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` (1500 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.

4.1.4.2.11 Configure graceful upgrade for TiCDC cluster

Note:

- If the TiCDC version is earlier than v6.3.0, TiDB Operator forces an upgrade on TiCDC, which might cause replication latency increase.
- The feature is available since TiDB Operator v1.3.8.

During TiCDC upgrade, TiDB Operator drains all replication workloads from TiCDC Pod before restarting TiCDC Pod. Only after the draining is completed or the draining exceeds the specified timeout (10 minutes by default), TiCDC Pod is restarted. If TiCDC has fewer than 2 instances, TiDB Operator forces an upgrade without waiting for the timeout.

If the draining exceeds the specified timeout, restarting TiCDC Pod causes issues such as more replication latency. To avoid the issues, you can configure the timeout `spec.ticdc.gracefulShutdownTimeout` (10 minutes by default) to a larger value. For example:

```
spec:
  ticdc:
    gracefulShutdownTimeout: 100m
```

4.1.4.2.12 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`

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}
```

4.1.4.2.13 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`, and so on.

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 Google Cloud 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.

If TiProxy is specified, `tiproxy-api` and `tiproxy-sql` services are also automatically created for use.

4.1.4.2.14 IPv6 Support

Starting v6.5.1, TiDB supports using IPv6 addresses for all network connections. If you deploy TiDB using TiDB Operator v1.4.3 or later versions, you can enable the TiDB cluster to listen on IPv6 addresses by configuring `spec.preferIPv6` to `true`.

```
spec:
  preferIPv6: true
  # ...
```

> **Warning:**
>
> This configuration can only be applied when deploying the TiDB cluster and cannot be enabled on deployed clusters, as it may cause the cluster to become unavailable.

4.1.4.3 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.

4.1.4.3.1 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}
```

Use topologySpreadConstraints to make pods evenly spread

By configuring `topologySpreadConstraints`, you can make pods evenly spread in different topologies. For instructions about configuring `topologySpreadConstraints`, see Pod Topology Spread Constraints.

You can either configure `topologySpreadConstraints` at a cluster level (`spec.topologySpreadConstraints`) for all components or at a component level (such as `spec.tidb.topologySpreadConstraints`) for specific components.

The following is an example configuration:

```
topologySpreadConstraints:
- topologyKey: kubernetes.io/hostname
- topologyKey: topology.kubernetes.io/zone
```

The example configuration can make pods of the same component evenly spread on different zones and nodes.

Currently, `topologySpreadConstraints` only supports the configuration of the `topologyKey` field. In the pod spec, the above example configuration will be automatically expanded as follows:

```
topologySpreadConstraints:
- topologyKey: kubernetes.io/hostname
  maxSkew: 1
  whenUnsatisfiable: DoNotSchedule
```

```
  labelSelector: <object>
- topologyKey: topology.kubernetes.io/zone
  maxSkew: 1
  whenUnsatisfiable: DoNotSchedule
  labelSelector: <object>
```

4.1.4.3.2 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 on Kubernetes:

- 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.

- 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`.

- Set the topological information of the Node where the TiDB node is located.

  Since TiDB Operator v1.4.0, if the deployed TiDB version >= v6.3.0, TiDB Operator automatically obtains the topological information of the Node for TiDB and calls the

corresponding interface of the TiDB server to set this information as TiDB's labels. Based on these labels, TiDB sends the Follower Read requests to the correct replicas.

Currently, TiDB Operator automatically sets the labels for the TiDB server corresponding to the `location-labels` in `pd.config`. TiDB depends on the `zone` label to support some features of Follower Read. TiDB Operator obtains the value of `zone`, `failure-domain.beta.kubernetes.io/zone`, and `topology.kubernetes.io/zone` labels as `zone`. TiDB Operator only sets labels of the node where the TiDB server is located and ignores other labels.

- Set the topological information of the Node where the TiProxy node is located.

  Starting from TiDB Operator v1.6.0, if the deployed TiProxy version >= v1.1.0, TiDB Operator automatically obtains the topological information of the Node for TiProxy and calls the corresponding interface of the TiProxy to set this information as TiProxy's labels. Based on these labels, TiProxy prioritizes forwarding requests to a local TiDB server.

  Currently, TiDB Operator automatically sets the labels for the TiProxy node corresponding to the `location-labels` in `pd.config`. TiProxy depends on the `zone` label to forward requests to a local TiDB server. TiDB Operator obtains the value of `zone`, `failure-domain.beta.kubernetes.io/zone`, and `topology.kubernetes.io/zone` labels as `zone`. TiDB Operator only sets labels of the node where the TiProxy is located and ignores other labels.

Starting from v1.4.0, when setting labels for TiKV and TiDB nodes, TiDB Operator supports setting shortened aliases for some labels provided by Kubernetes by default. In some scenarios, using aliases can help optimize the scheduling performance of PD. When you use TiDB Operator to set aliases for the `location-labels` of PD, if there are no corresponding labels for a Kubernetes node, then TiDB Operator uses the original labels automatically.

Currently, TiDB Operator supports the following label aliases:

- `region`: corresponds to `topology.kubernetes.io/region` and `failure-domain.beta.kubernetes.io/region`.
- `zone`: corresponds to `topology.kubernetes.io/zone` and `failure-domain.beta.kubernetes.io/zone`.
- `host`: corresponds to `kubernetes.io/hostname`.

For example, if labels such as `region`, `zone`, and `host` are not set on each node of Kubernetes, setting the `location-labels` of PD as `["topology.kubernetes.io/region", "topology.kubernetes.io/zone", "kubernetes.io/hostname"]` is the same as `["region", "zone", "host"]`.

4.1.5 Deploy TiDB on General Kubernetes

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

4.1.5.1 Prerequisites

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

4.1.5.2 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 v8.5.0):

```
pingcap/pd:v8.5.0
pingcap/tikv:v8.5.0
pingcap/tidb:v8.5.0
pingcap/ticdc:v8.5.0
pingcap/tiflash:v8.5.0
pingcap/tiproxy:latest
pingcap/tidb-monitor-reloader:v1.0.1
pingcap/tidb-monitor-initializer:v8.5.0
```

```
grafana/grafana:7.5.11
prom/prometheus:v2.18.1
busybox:1.26.2
```

Next, download all these images with the following command:

```
docker pull pingcap/pd:v8.5.0
docker pull pingcap/tikv:v8.5.0
docker pull pingcap/tidb:v8.5.0
docker pull pingcap/ticdc:v8.5.0
docker pull pingcap/tiflash:v8.5.0
docker pull pingcap/tiproxy:latest
docker pull pingcap/tidb-monitor-reloader:v1.0.1
docker pull pingcap/tidb-monitor-initializer:v8.5.0
docker pull grafana/grafana:7.5.11
docker pull prom/prometheus:v2.18.1
docker pull busybox:1.26.2

docker save -o pd-v8.5.0.tar pingcap/pd:v8.5.0
docker save -o tikv-v8.5.0.tar pingcap/tikv:v8.5.0
docker save -o tidb-v8.5.0.tar pingcap/tidb:v8.5.0
docker save -o ticdc-v8.5.0.tar pingcap/ticdc:v8.5.0
docker save -o tiproxy-latest.tar pingcap/tiproxy:latest
docker save -o tiflash-v8.5.0.tar pingcap/tiflash:v8.5.0
docker save -o tidb-monitor-reloader-v1.0.1.tar pingcap/tidb-monitor-
   ↪ reloader:v1.0.1
docker save -o tidb-monitor-initializer-v8.5.0.tar pingcap/tidb-monitor
   ↪ -initializer:v8.5.0
docker save -o grafana-6.0.1.tar grafana/grafana:7.5.11
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-v8.5.0.tar
docker load -i tikv-v8.5.0.tar
docker load -i tidb-v8.5.0.tar
docker load -i ticdc-v8.5.0.tar
docker load -i tiproxy-latest.tar
docker load -i tiflash-v8.5.0.tar
docker load -i tidb-monitor-reloader-v1.0.1.tar
docker load -i tidb-monitor-initializer-v8.5.0.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.

4.1.5.3 Initialize the TiDB cluster

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

> **Note:**
>
> By default, TiDB (versions starting from v4.0.2 and released before February 20, 2023) 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. Starting from February 20, 2023, the telemetry feature is disabled by default in newly released TiDB versions. See TiDB Release Timeline for details.

4.1.5.4 Configure TiDB monitoring

For more information, see Deploy monitoring and alerts for a TiDB cluster.

> **Note:**
>
> TiDB monitoring does not persist data by default. To ensure long-term data availability, it is recommended to persist monitoring data. TiDB monitoring

does not include Pod CPU, memory, or disk monitoring, nor does it have an alerting system. For more comprehensive monitoring and alerting, it is recommended to Set kube-prometheus and AlertManager.

4.1.5.5 Collect logs

System and application logs can be useful for troubleshooting issues and automating operations. By default, TiDB components output logs to the container's `stdout` and `stderr`, and log rotation is automatically performed based on the container runtime environment. When a Pod restarts, container logs will be lost. To prevent log loss, it is recommended to Collect logs of TiDB and its related components.

4.1.6 Initialize a TiDB Cluster on Kubernetes

This document describes how to initialize a TiDB cluster on 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.

4.1.6.1 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.

4.1.6.1.1 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}
```

4.1.6.1.2 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.

4.1.6.2 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.

4.1.6.3 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;
    GRAND 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.

4.1.6.4 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
```

4.1.7 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.

4.1.7.1 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

4.1.7.2 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}"
```

4.1.7.3 LoadBalancer

If the TiDB cluster runs in an environment with LoadBalancer, such as on Google Cloud 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 and GKE.

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

4.2 On Public Cloud Kubernetes

4.2.1 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 on General Kubernetes.

4.2.1.1 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.

4.2.1.2 Recommended instance types and storage

- Instance types: to gain better performance, the following is recommended:
  - PD nodes: `c7g.xlarge`
  - TiDB nodes: `c7g.4xlarge`
  - TiKV or TiFlash nodes: `m7g.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

4.2.1.3 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
addons:
  - name: aws-ebs-csi-driver

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

```
  availabilityZones: ["ap-northeast-1c"]
  instanceType: c5.2xlarge
  labels:
    dedicated: tidb
  taints:
    dedicated: tidb:NoSchedule
  iam:
    withAddonPolicies:
      ebs: true
- name: pd-1a
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1a"]
  instanceType: c7g.xlarge
  labels:
    dedicated: pd
  taints:
    dedicated: pd:NoSchedule
  iam:
    withAddonPolicies:
      ebs: true
- name: pd-1d
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  instanceType: c7g.xlarge
  labels:
    dedicated: pd
  taints:
    dedicated: pd:NoSchedule
  iam:
    withAddonPolicies:
      ebs: true
- name: pd-1c
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1c"]
  instanceType: c7g.xlarge
  labels:
    dedicated: pd
  taints:
    dedicated: pd:NoSchedule
  iam:
    withAddonPolicies:
      ebs: true
```

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

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.

4.2.1.4 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.

4.2.1.4.1 Configure `gp2`

> **note:**
>
> Starting from EKS Kubernetes 1.23, you need to deploy the EBS CSI driver before using the default gp2 storage class. For details, refer to the notice for Amazon EKS Kubernetes 1.23.

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.

4.2.1.4.2 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. Set ebs-csi-node `toleration`.

```
kubectl patch -n kube-system ds ebs-csi-node -p '{"spec":{"template":{"
    ↪ spec":{"tolerations":[{"operator":"Exists"}]}}}}'
```

   Expected output:

```
daemonset.apps/ebs-csi-node patched
```

3. 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
```

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

```
spec:
  tikv:
    ...
    storageClassName: gp3
```

5. 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.

4.2.1.4.3 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
  iam:
    withAddonPolicies:
      ebs: true
  ...
```

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`.

4.2.1.5 Deploy TiDB Operator

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

4.2.1.6 Deploy a TiDB cluster and the monitoring component

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

4.2.1.6.1 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`.

4.2.1.6.2 Deploy

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

```
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/aws/tidb-cluster.yaml && \
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/aws/tidb-monitor.yaml && \
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/aws/tidb-dashboard.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 "internal" to "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.

4.2.1.6.3 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
```

4.2.1.7 Access the database

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

4.2.1.7.1 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.

4.2.1.7.2 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 --comments -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 -n tidb-cluster`.

For example:

```
$ mysql --comments -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: 8.0.11-TiDB-v8.5.0 TiDB Server (Apache License 2.0) Community Edition, MySQL 8.0 compatible

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

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 (versions starting from v4.0.2 and released before February 20, 2023) 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. Starting from

February 20, 2023, the telemetry feature is disabled by default in newly released TiDB versions. See TiDB Release Timeline for details.

4.2.1.8 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`.

4.2.1.9 Access the TiDB Dashboard

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

4.2.1.10 Upgrade

To upgrade the TiDB cluster, execute the following command:

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

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

4.2.1.11 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.

4.2.1.11.1 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.

4.2.1.11.2 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.

4.2.1.12 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.

4.2.1.12.1 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
  iam:
    withAddonPolicies:
      ebs: true
- name: tiflash-1d
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  labels:
    dedicated: tiflash
  taints:
    dedicated: tiflash:NoSchedule
  iam:
    withAddonPolicies:
      ebs: true
- name: tiflash-1c
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1c"]
  labels:
    dedicated: tiflash
  taints:
    dedicated: tiflash:NoSchedule
  iam:
    withAddonPolicies:
      ebs: true
- name: ticdc-1a
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1a"]
  labels:
    dedicated: ticdc
  taints:
    dedicated: ticdc:NoSchedule
  iam:
    withAddonPolicies:
      ebs: true
- name: ticdc-1d
  desiredCapacity: 1
  privateNetworking: true
  availabilityZones: ["ap-northeast-1d"]
  labels:
    dedicated: ticdc
```

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

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.

4.2.1.12.2 Configure and deploy

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

```
spec:
  ...
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 0
    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.

4.2.1.13 Configure TiDB monitoring

For more information, see Deploy monitoring and alerts for a TiDB cluster.

> **Note:**
>
> TiDB monitoring does not persist data by default. To ensure long-term data availability, it is recommended to persist monitoring data. TiDB monitoring does not include Pod CPU, memory, or disk monitoring, nor does it have an alerting system. For more comprehensive monitoring and alerting, it is recommended to Set kube-prometheus and AlertManager.

4.2.1.14 Collect logs

System and application logs can be useful for troubleshooting issues and automating operations. By default, TiDB components output logs to the container's `stdout` and `stderr`, and log rotation is automatically performed based on the container runtime environment. When a Pod restarts, container logs will be lost. To prevent log loss, it is recommended to Collect logs of TiDB and its related components.

4.2.2 Deploy TiDB on Google Cloud GKE

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

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

4.2.2.1 Prerequisites

Before deploying a TiDB cluster on 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 Google Cloud 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

4.2.2.2 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.

4.2.2.3 Configure the Google Cloud service

Configure your Google Cloud project and default region:

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

4.2.2.4 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.

4.2.2.5 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`.

4.2.2.5.1 Use local storage

For the production environment, use zonal persistent disks.

If you need to simulate bare-metal performance, some Google Cloud 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/v1.6.1/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`.

4.2.2.6 Deploy TiDB Operator

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

4.2.2.7 Deploy a TiDB cluster and the monitoring component

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

4.2.2.7.1 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`.

4.2.2.7.2 Deploy

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

```
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/gcp/tidb-cluster.yaml && \
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/gcp/tidb-monitor.yaml && \
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/gcp/tidb-dashboard.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.

4.2.2.7.3 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
```

4.2.2.8 Access the TiDB database

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

4.2.2.8.1 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.

4.2.2.8.2 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 --comments -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 --comments -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: 8.0.11-TiDB-v8.5.0 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 8.0 compatible

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

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 (versions starting from v4.0.2 and released before February 20, 2023) 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. Starting from February 20, 2023, the telemetry feature is disabled by default in newly released TiDB versions. See TiDB Release Timeline for details.

4.2.2.8.3 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`.

4.2.2.8.4 Access TiDB Dashboard Web UI

Obtain the `LoadBalancer` domain name of TiDB Dashboard by running the following command:

```
kubectl -n tidb-cluster get svc basic-tidb-dashboard-exposed
```

The following is an example:

```
$ kubectl -n tidb-cluster get svc basic-tidb-dashboard-exposed
NAME                  TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)
    ↪             AGE
basic-tidb-dashboard-exposed       LoadBalancer 10.15.255.169
    ↪ 34.123.168.114 12333:30657/TCP     35m
```

You can view monitoring metrics of TiDB Dashboard by visiting `${EXTERNAL-IP}:12333` using your web browser.

4.2.2.9 Upgrade

To upgrade the TiDB cluster, execute the following command:

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

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

4.2.2.10 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.

4.2.2.10.1 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`.

4.2.2.10.2 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.

4.2.2.11 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.

4.2.2.11.1 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
```

4.2.2.11.2 Configure and deploy

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

```
spec:
  ...
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 0
    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.

4.2.2.12 Configure TiDB monitoring

For more information, see Deploy monitoring and alerts for a TiDB cluster.

> **Note:**
>
> TiDB monitoring does not persist data by default. To ensure long-term data availability, it is recommended to persist monitoring data. TiDB monitoring does not include Pod CPU, memory, or disk monitoring, nor does it have an alerting system. For more comprehensive monitoring and alerting, it is recommended to Set kube-prometheus and AlertManager.

4.2.2.13 Collect logs

System and application logs can be useful for troubleshooting issues and automating operations. By default, TiDB components output logs to the container's `stdout` and `stderr`, and log rotation is automatically performed based on the container runtime environment. When a Pod restarts, container logs will be lost. To prevent log loss, it is recommended to Collect logs of TiDB and its related components.

4.2.3 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 on General Kubernetes.

4.2.3.1 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.

4.2.3.2 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).

4.2.3.2.1 Create an AKS cluster with CSI enabled

To create an AKS cluster with CSI enabled, run 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
```

4.2.3.2.2 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 \
    --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 \
    --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 \
    --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 \
    --node-count 3 \
    --labels dedicated=tikv \
    --node-taints dedicated=tikv:NoSchedule \
    --enable-ultra-ssd
```

4.2.3.2.3 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 \
    --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 \
    --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 \
    --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.

4.2.3.3 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
```

4.2.3.4 Deploy TiDB Operator

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

4.2.3.5 Deploy a TiDB cluster and the monitoring component

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

4.2.3.5.1 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`.

4.2.3.5.2 Deploy

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

```
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/aks/tidb-cluster.yaml && \
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/aks/tidb-monitor.yaml && \
curl -O https://raw.githubusercontent.com/pingcap/tidb-operator/v1.6.1/
    ↪ examples/aks/tidb-dashboard.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.

4.2.3.5.3 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
```

4.2.3.6 Access the database

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

4.2.3.6.1 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.

4.2.3.6.2 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: 8.0.11-TiDB-v8.5.0 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 8.0 compatible

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

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 (versions starting from v4.0.2 and released before February 20, 2023) 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. Starting from February 20, 2023, the telemetry feature is disabled by default in newly released TiDB versions. See TiDB Release Timeline for details.

4.2.3.7 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`.

4.2.3.8 Access TiDB Dashboard

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

4.2.3.9 Upgrade

To upgrade the TiDB cluster, execute the following command:

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

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

4.2.3.10 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.

4.2.3.10.1 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`.

4.2.3.10.2 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.

4.2.3.11 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.

4.2.3.11.1 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 \
    --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 \
    --node-count 3 \
    --labels dedicated=ticdc \
    --node-taints dedicated=ticdc:NoSchedule
```

4.2.3.11.2 Configure and deploy

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

```
spec:
  ...
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 0
    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.

4.2.3.12 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:
    ...
    storageClassName: ultra
```

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.

4.2.3.13 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 \
    --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/v1.6.1/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

4.2.3.14 Configure TiDB monitoring

For more information, see Deploy monitoring and alerts for a TiDB cluster.

> **Note:**
>
> TiDB monitoring does not persist data by default. To ensure long-term data availability, it is recommended to persist monitoring data. TiDB monitoring does not include Pod CPU, memory, or disk monitoring, nor does it have an alerting system. For more comprehensive monitoring and alerting, it is recommended to Set kube-prometheus and AlertManager.

4.2.3.15 Collect logs

System and application logs can be useful for troubleshooting issues and automating operations. By default, TiDB components output logs to the container's `stdout` and `stderr`, and log rotation is automatically performed based on the container runtime environment. When a Pod restarts, container logs will be lost. To prevent log loss, it is recommended to Collect logs of TiDB and its related components.

4.3 Deploy a TiDB Cluster on ARM64 Machines

This document describes how to deploy a TiDB cluster on ARM64 machines (including AWS Graviton instances).

4.3.1 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.

4.3.2 Deploy TiDB operator

- If your TiDB operator is v1.3.1 or later, deploy TiDB Operator normally. You don't need to do the following to change images.
- If your TiDB operator is earlier than v1.3.1, the process of deploying TiDB operator on ARM64 machines is the same as the process of Deploy TiDB Operator on Kubernetes. The only difference is that, you should change 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.

```
yaml  # ...  operatorImage: pingcap/tidb-operator-arm64:v1.3.1 # ...
↪ tidbBackupManagerImage: pingcap/tidb-backup-manager-arm64:v1.3.1 # ...
```

4.3.3 Deploy a TiDB cluster

- If your TiDB cluster is v5.4.2 or later, deploy the TiDB cluster normally. You don't need to do the following to change images.
- If your TiDB cluster is earlier than v5.4.2, 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.

```
yaml apiVersion: pingcap.com/v1alpha1 kind: TidbCluster metadata: name:
↪ ${cluster_name} namespace: ${cluster_namespace} spec: version: "v8.5.0"
↪ # ... 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 # ...
```

4.3.4 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 on 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
  # ...
```

4.3.5 Deploy monitoring for a TiDB cluster

- If your TiDB cluster is v5.4.2 or later, deploy monitoring and alerts normally. You don't need to do the following to change images.
- If your TiDB cluster is earlier than v5.4.2, 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` field in the TidbMonitor definition file to the ARM64 image.

```
yaml apiVersion: pingcap.com/v1alpha1 kind: TidbMonitor metadata: name:
↪ ${monitor_name} spec: # ... initializer: baseImage: pingcap/tidb
↪ -monitor-initializer-arm64 version: v5.4.1 # ...
```

4.4 Deploy the HTAP Storage Engine Tiflash for an Existing TiDB Cluster

This document describes how to add or remove the TiDB HTAP storage engine TiFlash for an existing TiDB cluster on Kubernetes. As a columnar storage extension of TiKV,

TiFlash provides both good isolation level and strong consistency guarantee.

> **Note:**
>
> If a TiDB cluster has not been deployed yet, instead of referring to this document, you can configure a TiDB cluster on Kubernetes with the TiFlash-related parameters, and then deploy the TiDB cluster.

4.4.1 Usage scenarios

This document is applicable to scenarios in which you already have a TiDB cluster and need to use TiDB HTAP capabilities by deploying TiFlash, such as the following:

- Hybrid workload scenarios with online real-time analytic processing
- Real-time stream processing scenarios
- Data hub scenarios

4.4.2 Deploy TiFlash

If you need to deploy TiFlash for an existing TiDB cluster, do the following:

> **Note:**
>
> If your server does not have an external network, you can download the required Docker image on the machine with an external network, upload the Docker image to your server, and then use `docker load` to install the Docker image on the server. For details, see deploy the TiDB cluster.

1. Edit the `TidbCluster` Custom Resource (CR):

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

2. Add the TiFlash configuration as the following example:

```
spec:
tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 0
    replicas: 1
```

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

3. 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 `requests.storage` and `storageClassName`. For example:

```
tiflash:
  baseImage: pingcap/tiflash
  maxFailoverCount: 0
  replicas: 1
  storageClaims:
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
  - resources:
      requests:
        storage: 100Gi
    storageClassName: local-storage
```

Note:

- When deploying TiFlash for the first time, it is recommended that you plan how many PVs are required and configure the number of `resources` items in `storageClaims` accordingly.
- Once the deployment of TiFlash is completed, if you need to mount additional PVs for TiFlash, updating `storageClaims` directly to add disks does not take effect. This is because TiDB Operator manages TiFlash by creating a StatefulSet, and the `StatefulSet` does not support modifying `volumeClaimTemplates` after being created.

4. Configure the relevant parameters of `spec.tiflash.config` in TidbCluster CR. For example:

```
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 more TiFlash parameters that can be configured, refer to TiFlash Configuration Documentation.

Note:

For different TiFlash versions, note the following configuration differences:

- If TiFlash version <= v4.0.4, you need to set `spec.tiflash.config.config.flash.service_addr` to `${clusterName}-tiflash-POD_NUM.${clusterName}-tiflash-peer.${namespace}.svc:3930` in TidbCluster CR, where `${clusterName}` and `${namespace}` need to be replaced according to the real case.
- If TiFlash version >= v4.0.5, there is no need to manually configure `spec.tiflash.config.config.flash.service_addr`.
- If you upgrade from TiFlash v4.0.4 or an earlier version to TiFlash v4.0.5 or a later version, you need to delete the configuration of `spec.tiflash.config.config.flash.service_addr` from the `TidbCluster` CR.

4.4.3 Adding PVs to TiFlash

Once the deployment of TiFlash is completed, to add PVs for TiFlash, you need to update the `storageClaims` to add disks, and then manually delete the TiFlash StatefulSet. The following are the detailed steps.

Warning:

Deleting the TiFlash StatefulSet makes the TiFlash cluster unavailable during the deletion and affects related business. You must be cautious about whether to do the following.

1. Edit the TidbCluster Custom Resource (CR).

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

2. TiDB Operator automatically mounts PVs in the **order** of the items in the `storageClaims` list. If you need to add more `resources` items to TiFlash, make sure to append new items only to the **end** of the original items, and **DO NOT** modify the order of the original items. For example:

```
tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 0
    replicas: 1
    storageClaims:
    - resources:
        requests:
            storage: 100Gi
      storageClassName: local-storage
    - resources:
        requests:
            storage: 100Gi
      storageClassName: local-storage
    - resources: #newly added
        requests: #newly added
            storage: 100Gi #newly added
      storageClassName: local-storage #newly added
```

3. Manually delete the TiFlash StatefulSet by running the following command. Then, wait for the TiDB Operator to recreate the TiFlash StatefulSet.

```
kubectl delete sts -n ${namespace} ${cluster_name}-tiflash
```

4.4.4 Remove TiFlash

If your TiDB cluster no longer needs the TiDB HTAP storage engine TiFlash, take the following steps to 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 on 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 patch tidbcluster ${cluster_name} -n ${namespace} --type merge
    ↪ -p '{"spec":{"tiflash":{"replicas": 0}}}'
```

4. Check the state of TiFlash Pods and TiFlash stores.

   1. Run the following command to check whether you delete the TiFlash Pod successfully:

```
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.

5. 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.

6. Delete the TiFlash StatefulSet.

7. To modify the TidbCluster CR and delete the `spec.tiflash` field, run the following command:

```shell
kubectl patch tidbcluster ${cluster_name} -n ${namespace} --type json -p '[{"op":"remove", "path":"/spec/tiflash"}]'
```

8. 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}
```

9. 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.

10. (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.

```
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:

```
kubectl get pv -l app.kubernetes.io/component=tiflash,app.
    ↪ kubernetes.io/instance=${cluster_name}
```

4.5 Deploy TiProxy Load Balancer for an Existing TiDB Cluster

This topic describes how to deploy or remove the TiDB load balancer TiProxy for an existing TiDB cluster on Kubernetes. TiProxy is placed between the client and TiDB server to provide load balancing, connection persistence, and service discovery for TiDB.

> **Note:**
>
> If you have not deployed a TiDB cluster, you can add TiProxy configurations when configuring a TiDB cluster and then deploy a TiDB cluster. In that case, you do not need to refer to this topic.

4.5.1 Deploy TiProxy

If you need to deploy TiProxy for an existing TiDB cluster, follow these steps:

Note:

If your server does not have access to the internet, refer to Deploy a TiDB Cluster to download the `pingcap/tiproxy` Docker image to a machine with access to the internet and then upload the Docker image to your server. Then, use `docker load` to install the Docker image on your server.

1. Edit the TidbCluster Custom Resource (CR):

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

2. Add the TiProxy configuration as shown in the following example:

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

3. Configure the related parameters in `spec.tiproxy.config` of the TidbCluster CR. For example:

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

   For more information about TiProxy configuration, see TiProxy Configuration.

4. Configure the related parameters in `spec.tidb` of the TidbCluster CR. For example:

   - It is recommended to configure `graceful-wait-before-shutdown` to a value greater than the maximum duration of the transactions in your application. This is used together with TiProxy's connection migration feature. For more information, see TiProxy Limitations.

     `yaml spec: tidb: config: | graceful-wait-before-shutdown = 30`

   - If TLS is enabled for the cluster, skip this step. If TLS is not enabled for the cluster, you need to generate a self-signed certificate and manually configure `session-token-signing-cert` and `session-token-signing-key` for TiDB:

```
spec:
  tidb:
    additionalVolumes:
```

```
    - name: sessioncert
      secret:
        secretName: sessioncert-secret
  additionalVolumeMounts:
    - name: sessioncert
      mountPath: /var/session
  config: |
    session-token-signing-cert = "/var/session/tls.crt"
    session-token-signing-key = "/var/session/tls.key"
```

For more information, see session-token-signing-cert.

After TiProxy is started, you can find the corresponding `tiproxy-sql` load balancer service by running the following command.

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

4.5.2 Remove TiProxy

If your TiDB cluster no longer needs TiProxy, follow these steps to remove it.

1. Modify `spec.tiproxy.replicas` to `0` to remove the TiProxy Pod by running the following command.

```
kubectl patch tidbcluster ${cluster_name} -n ${namespace} --type merge
    ↪ -p '{"spec":{"tiproxy":{"replicas": 0}}}'
```

2. Check the status of the TiProxy Pod.

```
kubectl get pod -n ${namespace} -l app.kubernetes.io/component=tiproxy,
    ↪ app.kubernetes.io/instance=${cluster_name}
```

   If the output is empty, the TiProxy Pod has been successfully removed.

3. Delete the TiProxy StatefulSet.

   1. Modify the TidbCluster CR and delete the `spec.tiproxy` field by running the following command:

```
kubectl patch tidbcluster ${cluster_name} -n ${namespace} --type
    ↪ json -p '[{"op":"remove", "path":"/spec/tiproxy"}]'
```

   2. Delete the TiProxy StatefulSet by running the following command:

```
kubectl delete statefulsets -n ${namespace} -l app.kubernetes.io/
    ↪ component=tiproxy,app.kubernetes.io/instance=${cluster_name}
```

3. Check whether the TiProxy StatefulSet has been successfully deleted by running the following command:

```
kubectl get sts -n ${namespace} -l app.kubernetes.io/component=
    ↪ tiproxy,app.kubernetes.io/instance=${cluster_name}
```

If the output is empty, the TiProxy StatefulSet has been successfully deleted.

4.6 Deploy TiDB Across Multiple Kubernetes Clusters

4.6.1 Build Multiple Interconnected AWS EKS Clusters

This document describes how to create multiple AWS EKS clusters and configure network peering between these clusters. These interconnected clusters can be used for deploying TiDB clusters across multiple Kubernetes clusters. The example in this document shows how to configure three-cluster network peering.

If you need to deploy TiDB on a single AWS EKS cluster, refer to Deploy TiDB on AWS EKS.

4.6.1.1 Prerequisites

Before you deploy EKS clusters, make sure you have completed the following preparations:

- Install Helm 3. You need to use Helm to install TiDB Operator.
- Complete all steps in Getting started with Amazon EKS—eksctl.

  This tutorial includes the following tasks:

  - Install and configure the AWS CLI (`awscli`).
  - Install and configure the CLI for creating Kubernetes clusters (`eksctl`).
  - Install the Kubernetes CLI (`kubectl`)

- Grant AWS Access Key the minimum permissions required for `eksctl` and the permissions required for creating a Linux bastion.

To verify whether you have correctly configured the AWS CLI, run the `aws configure list` command. If the output shows the values of `access_key` and `secret_key`, you have successfully configured the AWS CLI. Otherwise, you need to reconfigure the AWS CLI.

4.6.1.2 Step 1. Start the Kubernetes cluster

Define the configuration files of three EKS clusters as `cluster_1.yaml`, `cluster_2.yaml`, and `cluster_3.yaml`, and create three clusters using `eksctl`.

1. Define the configuration file of cluster 1, and create cluster 1.

    1. Save the following content as `cluster_1.yaml`. `${cluster_1}` is the name of the EKS cluster. `${region_1}` is the Region that the EKS cluster is deployed in. `${cidr_block_1}` is the CIDR block for the VPC that the EKS cluster is deployed in.

    ```
    apiVersion: eksctl.io/v1alpha5
    kind: ClusterConfig

    metadata:
      name: ${cluster_1}
      region: ${region_1}

    # nodeGroups ...

    vpc:
      cidr: ${cidr_block_1}
    ```

    For the configuration of the `nodeGroups` field, refer to Create an EKS cluster and a node pool.

    2. Create cluster 1 by running the following command:

    ```
    eksctl create cluster -f cluster_1.yaml
    ```

    After running the command above, wait until the EKS cluster is successfully created and the node group is created and added to the EKS cluster. This process might take 5 to 20 minutes. For more cluster configuration, refer to Using Config Files.

2. Follow the instructions in the previous step and create cluster 2 and cluster 3.

    The CIDR block for each EKS cluster **must not** overlap with that of each other.

    In the following sections:

    - `${cluster_1}`, `${cluster_2}`, and `${cluster_3}` refer to the cluster names.
    - `${region_1}`, `${region_2}`, and `${region_3}` refer to the Regions that the clusters are deployed in.
    - `${cidr_block_1}`, `${cidr_block_2}`, and `${cidr_block_3}` refer to the CIDR blocks for the VPCs that the clusters are deployed in.

3. After the clusters are created, obtain the Kubernetes context of each cluster. The contexts are used in the subsequent `kubectl` commands.

```
kubectl config get-contexts
```

Expected output

The context is in the NAME column.

In the following sections, `${context_1}`, `${context_2}`, and `${context_3}` refer to the context of each cluster.

4.6.1.3 Step 2. Configure the network

4.6.1.3.1 Set up VPC peering

To allow the three clusters to access each other, you need to create a VPC peering connection between the VPCs of every two clusters. For details on VPC peering, see AWS documentation.

1. Get the VPC ID of each cluster.

   The following example gets the VPC ID of cluster 1:

   ```
   eksctl get cluster ${cluster_1} --region ${region_1}
   ```

   Expected output

   The VPC ID is in the VPC column.

   In the following sections, `${vpc_id_1}`, `${vpc_id_2}`, and `${vpc_id_3}` refer to the VPC ID of each cluster.

2. Create a VPC peering connection between cluster 1 and cluster 2.

   1. Refer to AWS documentation and create a VPC peering. Use `${vpc_id_1}` as the requester VPC and `${vpc_id_2}` as the accepter VPC.
   2. Refer to AWS documentation and complete creating a VPC peering.

3. Follow the instructions in the previous step. Create a VPC peering connection between cluster 1 and cluster 3 and a VPC peering connection between cluster 2 and cluster 3.

4. Update the route tables for the VPC peering connection of the three clusters.

   You need to update the route tables of all subnets used by the clusters. Add two routes in each route table.

   The following example shows the route table of cluster 1:

Destination	Target	Status	Propagated
`${cidr_block_2}`	`${vpc_peering_id_12}`	Active	No
`${cidr_block_3}`	`${vpc_peering_id_13}`	Active	No

The **Destination** of each route is the CIDR block of another cluster. The **Target** is the VPC peering ID of the two clusters.

4.6.1.3.2 Update the security groups for the instances

1. Update the security group for cluster 1.

    1. Enter the AWS Security Groups Console and select the security group of cluster 1. The name of the security group is similar to `eksctl-${cluster_1}-cluster ↪ /ClusterSharedNodeSecurityGroup`.
    2. Add inbound rules to the security group to allow traffic from cluster 2 and cluster 3.

Type	Protocol	Port range	Source	Description
All traffic	All	All	Custom `${cidr_block_2}`	Allow cluster 2 to communicate with cluster 1
All traffic	All	All	Custom `${cidr_block_3}`	Allow cluster 3 to communicate with cluster 1

2. Follow the instructions in the previous step to update the security groups for cluster 2 and cluster 3.

4.6.1.3.3 Configure load balancers

Each cluster needs to expose its CoreDNS service to other clusters via a network load balancer. This section describes how to configure load balancers.

1. Create a load balancer service definition file `dns-lb.yaml` as follows:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: kube-dns
  name: across-cluster-dns-tcp
  namespace: kube-system
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-
       ↪ balancing-enabled: "true"
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
    service.beta.kubernetes.io/aws-load-balancer-internal: "true"
spec:
  ports:
```

```
  - name: dns
    port: 53
    protocol: TCP
    targetPort: 53
  selector:
    k8s-app: kube-dns
  type: LoadBalancer
```

2. Deploy the load balancer service in each cluster:

```
kubectl --context ${context_1} apply -f dns-lb.yaml

kubectl --context ${context_2} apply -f dns-lb.yaml

kubectl --context ${context_3} apply -f dns-lb.yaml
```

3. Get the load balancer name of each cluster, and wait for all load balancers to become `Active`.

   Get the load balancer names by running the following commands:

```
lb_name_1=$(kubectl --context ${context_1} -n kube-system get svc
   ↪ across-cluster-dns-tcp -o jsonpath="{.status.loadBalancer.ingress
   ↪ [0].hostname}" | cut -d - -f 1)

lb_name_2=$(kubectl --context ${context_2} -n kube-system get svc
   ↪ across-cluster-dns-tcp -o jsonpath="{.status.loadBalancer.ingress
   ↪ [0].hostname}" | cut -d - -f 1)

lb_name_3=$(kubectl --context ${context_3} -n kube-system get svc
   ↪ across-cluster-dns-tcp -o jsonpath="{.status.loadBalancer.ingress
   ↪ [0].hostname}" | cut -d - -f 1)
```

   Check the load balancer status of each cluster by running the following commands. If the output of all commands is `active`, all load balancers are in the `Active` state.

```
aws elbv2 describe-load-balancers --names ${lb_name_1} --region ${
   ↪ region_1} --query 'LoadBalancers[*].State' --output text

aws elbv2 describe-load-balancers --names ${lb_name_2} --region ${
   ↪ region_2} --query 'LoadBalancers[*].State' --output text

aws elbv2 describe-load-balancers --names ${lb_name_3} --region ${
   ↪ region_3} --query 'LoadBalancers[*].State' --output text
```

   Expected output

4. Check the IP address associated with the load balancer of each cluster.

   Check the IP address associated with the load balancer of cluster 1:

```
aws ec2 describe-network-interfaces --region ${region_1} --filters Name
    ↪ =description,Values="ELB net/${lb_name_1}*" --query '
    ↪ NetworkInterfaces[*].PrivateIpAddress' --output text
```

   Expected output

   Repeat the same step for cluster 2 and cluster 3.

   In the following sections, `${lb_ip_list_1}`, `${lb_ip_list_2}`, and `${lb_ip_list_3}` refer to the IP addresses associated with the load balancer of each cluster.

   The load balancers in different Regions might have different numbers of IP addresses. For example, in the example above, `${lb_ip_list_1}` is `10.1.175.233 10.1.144.196` ↪ .

4.6.1.3.4 Configure CoreDNS

To allow Pods in a cluster to access services in other clusters, you need to configure CoreDNS for each cluster to forward DNS requests to the CoreDNS services of other clusters.

You can configure CoreDNS by modifying the ConfigMap corresponding to the CoreDNS. For information on more configuration items, refer to Customizing DNS Service.

1. Modify the CoreDNS configuration of cluster 1.

   1. Back up the current CoreDNS configuration:

```
kubectl --context ${context_1} -n kube-system get configmap coredns
    ↪  -o yaml > ${cluster_1}-coredns.yaml.bk
```

   2. Modify the ConfigMap:

```
kubectl --context ${context_1} -n kube-system edit configmap
    ↪ coredns
```

      Modify the `data.Corefile` field as follows. In the example below, `${namespace_2` ↪ `}` and `${namespace_3}` are the namespaces that cluster 2 and cluster 3 deploy `TidbCluster` in.

> **Warning:**
> Because you cannot modify the cluster domain of an EKS cluster, you need to use the namespace as an identifier for DNS forwarding. Therefore, `${namespace_1}`, `${namespace_2}`, and `${namespace_3}` must be different from each other.

```
apiVersion: v1
kind: ConfigMap
# ...
data:
  Corefile: |
    .:53 {
       # Do not modify the default configuration.
    }
    ${namespace_2}.svc.cluster.local:53 {
        errors
        cache 30
        forward . ${lb_ip_list_2} {
            force_tcp
        }
    }
    ${namespace_3}.svc.cluster.local:53 {
        errors
        cache 30
        forward . ${lb_ip_list_3} {
            force_tcp
        }
    }
```

3. Wait for the CoreDNS to reload the configuration. It might take around 30 seconds.

2. Follow the instructions in the previous step, and modify the CoreDNS configuration of cluster 2 and cluster 3.

   For the CoreDNS configuration of each cluster, you need to perform the following operations:

   - Configure `${namespace_2}` and `${namespace_3}` to the namespace that the other two clusters deploy `TidbCluster` in.
   - Configure the IP address to the IP addresses of the load balancers of the other two clusters.

In the following sections, `${namespace_1}`, `${namespace_2}`, and `${namespace_3}` refer to the namespaces that each cluster deploy `TidbCluster` in.

4.6.1.4 Step 3. Verify the network interconnectivity

Before you deploy the TiDB cluster, you need to verify that the network between the EKS clusters is interconnected.

1. Save the following content in the `sample-nginx.yaml` file.

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-nginx
  labels:
    app: sample-nginx
spec:
  hostname: sample-nginx
  subdomain: sample-nginx-peer
  containers:
  - image: nginx:1.21.5
    imagePullPolicy: IfNotPresent
    name: nginx
    ports:
      - name: http
        containerPort: 80
  restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: sample-nginx-peer
spec:
  ports:
    - port: 80
  selector:
    app: sample-nginx
  clusterIP: None
```

2. Deploy the nginx service to the namespaces of the three clusters:

```
kubectl --context ${context_1} -n ${namespace_1} apply -f sample-nginx.
    ↪ yaml

kubectl --context ${context_2} -n ${namespace_2} apply -f sample-nginx.
    ↪ yaml

kubectl --context ${context_3} -n ${namespace_3} apply -f sample-nginx.
    ↪ yaml
```

3. Access the nginx services of each cluster to verify the network interconnectivity.

   The following command verifies the network from cluster 1 to cluster 2:

```
kubectl --context ${context_1} -n ${namespace_1} exec sample-nginx --
    ↪ curl http://sample-nginx.sample-nginx-peer.${namespace_2}.svc.
    ↪ cluster.local:80
```

If the output is the welcome page of nginx, the network is connected.

4. After the verification, delete the nginx services:

```
kubectl --context ${context_1} -n ${namespace_1} delete -f sample-nginx
    ↪ .yaml

kubectl --context ${context_2} -n ${namespace_2} delete -f sample-nginx
    ↪ .yaml

kubectl --context ${context_3} -n ${namespace_3} delete -f sample-nginx
    ↪ .yaml
```

4.6.1.5 Step 4. Deploy TiDB Operator

The `TidbCluster` CR of each cluster is managed by TiDB Operator of the cluster. Therefore, you must deploy TiDB Operator for each cluster.

Refer to Deploy TiDB Operator and deploy TiDB Operator in each EKS cluster. Note that you need to use `kubectl --context ${context}` and `helm --kube-context ${context}` in the commands to deploy TiDB Operator for each EKS cluster.

4.6.1.6 Step 5. Deploy TiDB clusters

Refer to Deploy a TiDB Cluster across Multiple Kubernetes Clusters and deploy a `TidbCluster` CR for each EKS cluster. Note the following operations:

- You must deploy the TidbCluster CR in the corresponding namespace configured in the Configure CoreDNS section. Otherwise, the TiDB cluster will fail to start.
- The cluster domain of each cluster must be set to "cluster.local".

Take cluster 1 as an example. When you deploy the `TidbCluster` CR to cluster 1, specify `metadata.namespace` as `${namespace_1}`:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${tc_name_1}
  namespace: ${namespace_1}
spec:
  # ...
```

```
  clusterDomain: "cluster.local"
  acrossK8s: true
```

4.6.1.7 What's next

- Read Deploy a TiDB Cluster across Multiple Kubernetes Clusters to learn how to manage a TiDB cluster across multiple Kubernetes clusters.

4.6.2 Build Multiple Interconnected Google Cloud GKE Clusters

This document describes how to create multiple Google Kubernetes Engine (GKE) clusters and configure network peering between these clusters. These interconnected clusters can be used for deploying TiDB clusters across multiple Kubernetes clusters. The example in this document shows how to configure three-cluster network peering.

If you need to deploy TiDB on a single GKE cluster, refer to Deploy TiDB on Google Cloud GKE.

4.6.2.1 Prerequisites

Before you deploy GKE clusters, make sure you have completed the following preparations:

- Install Helm 3. You need to use Helm to install TiDB Operator.
- Install gcloud: `gcloud` is the CLI for creating and managing Google Cloud services
- Complete the *Before you begin* section in GKE Quickstart.

4.6.2.2 Configure Google Cloud service

Configure your Google Cloud project by running the following command:

```
gcloud config set core/project <google-cloud-project>
```

4.6.2.3 Step 1. Create a VPC network

1. Create a VPC network with custom subnets:

   ```
   gcloud compute networks create ${network_name} --subnet-mode=custom
   ```

2. In the VPC network created above, create three subnets that belong to different regions. The CIDR block of each subnet does not overlap with that of each other.

```
gcloud compute networks subnets create ${subnet_1} \
    --region=${region_1} \
    --network=${network_name} \
    --range=10.0.0.0/16 \
    --secondary-range pods=10.10.0.0/16,services=10.100.0.0/16
```

```
gcloud compute networks subnets create ${subnet_2} \
    --region=${region_2} \
    --network=${network_name} \
    --range=10.1.0.0/16 \
    --secondary-range pods=10.11.0.0/16,services=10.101.0.0/16
```

```
gcloud compute networks subnets create ${subnet_3} \
    --region=${region_3} \
    --network=${network_name} \
    --range=10.2.0.0/16 \
    --secondary-range pods=10.12.0.0/16,services=10.102.0.0/16
```

`${subnet_1}`, `${subnet_2}`, and `${subnet_3}` refer to the names of the three subnets.

`--range=10.0.0.0/16` specifies the CIDR block of the `${subnet_1}` in the cluster. The CIDR blocks of all cluster subnets **must not** overlap with each other.

`--secondary-range pods=10.11.0.0/16,services=10.101.0.0/16` specifies the CIRD block used by Kubernetes Pods and Services. This CIRD block will be used later.

4.6.2.4 Step 2. Start the Kubernetes cluster

Create three GKE clusters, and each cluster uses one of the subnets created in Step 1.

1. Create three GKE clusters. Each cluster has a default node pool.

```
gcloud beta container clusters create ${cluster_1} \
    --region ${region_1} --num-nodes 1 \
    --network ${network_name} --subnetwork ${subnet_1} \
    --cluster-dns clouddns --cluster-dns-scope vpc \
    --cluster-dns-domain ${cluster_domain_1} \
    --enable-ip-alias \
    --cluster-secondary-range-name=pods --services-secondary-range-name
        ↪ =services
```

```
gcloud beta container clusters create ${cluster_2} \
    --region ${region_2} --num-nodes 1 \
    --network ${network_name} --subnetwork ${subnet_2} \
    --cluster-dns clouddns --cluster-dns-scope vpc \
```

```
    --cluster-dns-domain ${cluster_domain_2} \
    --enable-ip-alias \
    --cluster-secondary-range-name=pods --services-secondary-range-name
        ↪ =services
```

```
gcloud beta container clusters create ${cluster_3} \
    --region ${region_3} --num-nodes 1 \
    --network ${network_name} --subnetwork ${subnet_3} \
    --cluster-dns clouddns --cluster-dns-scope vpc \
    --cluster-dns-domain ${cluster_domain_3} \
    --enable-ip-alias \
    --cluster-secondary-range-name=pods --services-secondary-range-name
        ↪ =services
```

In the commands above, `${cluster_domain_n}` refers to the domain name of the `n`th cluster. In the following deployment steps, you need to configure `spec.clusterDomain` in TidbCluster CR to `${cluster_domain_n}`.

In the commands above, the Cloud DNS in VPC scope is used so that the cluster can parse the Pod and Service addresses in other clusters.

2. Create the dedicated node pools used by PD, TiKV, and TiDB for each cluster.

   Take cluster 1 as an example:

```
gcloud container node-pools create pd --cluster ${cluster_1} --machine-
    ↪ type n1-standard-4 --num-nodes=1 \
    --node-labels=dedicated=pd --node-taints=dedicated=pd:NoSchedule
gcloud container node-pools create tikv --cluster ${cluster_1} --
    ↪ machine-type n1-highmem-8 --num-nodes=1 \
    --node-labels=dedicated=tikv --node-taints=dedicated=tikv:
        ↪ NoSchedule
gcloud container node-pools create tidb --cluster ${cluster_1} --
    ↪ machine-type n1-standard-8 --num-nodes=1 \
    --node-labels=dedicated=tidb --node-taints=dedicated=tidb:
        ↪ NoSchedule
```

3. Obtain the Kubernetes context of each cluster. The context will be used in the subsequent `kubectl` commands.

```
kubectl config get-contexts
```

   The expected output is as follows. The context is in the `NAME` column.

```
CURRENT  NAME                          CLUSTER                       AUTHINFO
    ↪                          NAMESPACE
*        gke_pingcap_us-west1_tidb-1 gke_pingcap_us-west1_tidb-1
    ↪ gke_pingcap_us-west1_tidb-1
```

```
gke_pingcap_us-west2_tidb-2 gke_pingcap_us-west2_tidb-2
    ↪ gke_pingcap_us-west2_tidb-2
gke_pingcap_us-west3_tidb-3 gke_pingcap_us-west3_tidb-3
    ↪ gke_pingcap_us-west3_tidb-3
```

In the following sections, `${context_1}`, `${context_2}`, and `${context_3}` refer to the context of each cluster.

4.6.2.4.1 Configure the firewall rules

1. Update the firewall rules for cluster 1.

    1. Obtain the name of the firewall rule used for communication between GKE Pods. The name of the firewall rule is similar to `gke-${cluster_1}-${hash}-all`.

    ```
    gcloud compute firewall-rules list --filter='name~gke-${cluster_1
        ↪ }-.*-all'
    ```

    The expected output is as follows. The rule name is in the `NAME` column.

    ```
    NAME                          NETWORK    DIRECTION PRIORITY ALLOW
        ↪                        DENY DISABLED
    gke-${cluster_1}-b8b48366-all ${network} INGRESS 1000   tcp,udp,
        ↪ icmp,esp,ah,sctp        False
    ```

    2. Update the source range of the firewall rule. Add the CIDR blocks of the Pod network of the other two clusters to the source range:

    ```
    gcloud compute firewall-rules update ${firewall_rule_name} --
        ↪ source-ranges 10.10.0.0/16,10.11.0.0/16,10.12.0.0/16
    ```

    Run the following command to check whether the firewall rule is successfully updated:

    ```
    gcloud compute firewall-rules describe ${firewall_rule_name}
    ```

2. Follow the same steps to update the firewall rules for cluster 2 and cluster 3.

4.6.2.5 Step 3. Verify the network interconnectivity

Before you deploy the TiDB cluster, you need to verify that the network between the GKE clusters is interconnected.

1. Save the following content in the `sample-nginx.yaml` file.

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-nginx
  labels:
    app: sample-nginx
spec:
  hostname: sample-nginx
  subdomain: sample-nginx-peer
  containers:
  - image: nginx:1.21.5
    imagePullPolicy: IfNotPresent
    name: nginx
    ports:
      - name: http
        containerPort: 80
  restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: sample-nginx-peer
spec:
  ports:
    - port: 80
  selector:
    app: sample-nginx
  clusterIP: None
```

2. Deploy the nginx service in the namespaces of three clusters.

```
kubectl --context ${context_1} -n default apply -f sample-nginx.yaml

kubectl --context ${context_2} -n default apply -f sample-nginx.yaml

kubectl --context ${context_3} -n default apply -f sample-nginx.yaml
```

3. Access the nginx services of each cluster to verify the network interconnectivity.

   The following command verifies the network from cluster 1 to cluster 2:

```
kubectl --context ${context_1} exec sample-nginx -- curl http://sample-
    ↪ nginx.sample-nginx-peer.default.svc.${cluster_domain_2}:80
```

   If the output is the welcome page of nginx, the network is connected.

4. After the verification, delete the nginx services:

```
kubectl --context ${context_1} -n default delete -f sample-nginx.yaml
kubectl --context ${context_2} -n default delete -f sample-nginx.yaml
kubectl --context ${context_3} -n default delete -f sample-nginx.yaml
```

4.6.2.6 Step 4. Deploy TiDB Operator

The `TidbCluster` CR of each cluster is managed by TiDB Operator of the cluster. Therefore, you must deploy TiDB Operator for each cluster.

Refer to Deploy TiDB Operator and deploy TiDB Operator in each GKE cluster. Note that you need to use `kubectl --context ${context}` and `helm --kube-context ${context}` in the commands to deploy TiDB Operator for each GKE cluster.

4.6.2.7 Step 5. Deploy TiDB clusters

Refer to Deploy a TiDB Cluster across Multiple Kubernetes Clusters, and deploy a `TidbCluster` CR for each GKE cluster.

In the `TidbCluster` CR, the `spec.clusterDomain` field must be the same as `${cluster_domain_n}` defined in Step 2.

For example, when you deploy the `TidbCluster` CR to cluster 1, specify `spec.clusterDomain` as `${cluster_domain_1}`:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
### ...
spec:
  #..
  clusterDomain: "${cluster_domain_1}"
  acrossK8s: true
```

4.6.2.8 What's next

- Read Deploy a TiDB Cluster across Multiple Kubernetes Clusters to learn how to manage a TiDB cluster across multiple Kubernetes clusters.

4.6.3 Deploy a TiDB Cluster across Multiple Kubernetes Clusters

To deploy a TiDB cluster across multiple Kubernetes clusters refers to deploying **one** TiDB cluster on multiple interconnected Kubernetes clusters. Each component of the cluster is distributed on multiple Kubernetes clusters to achieve disaster recovery among Kubernetes clusters. The interconnected network of Kubernetes clusters means that Pod IP can be accessed in any cluster and between clusters, and Pod FQDN records can be looked up by querying the DNS service in any cluster and between clusters.

4.6.3.1 Prerequisites

You need to configure the Kubernetes network and DNS so that the Kubernetes cluster meets the following conditions:

- The TiDB components on each Kubernetes cluster can access the Pod IP of all TiDB components in and between clusters.
- The TiDB components on each Kubernetes cluster can look up the Pod FQDN of all TiDB components in and between clusters.

To build multiple connected EKS or GKE clusters, refer to Build Multiple Interconnected AWS EKS Clusters or Build Multiple Interconnected Google Cloud GKE Clusters.

4.6.3.2 Supported scenarios

Currently supported scenarios:

- Deploy a new TiDB cluster across multiple Kubernetes clusters.
- Deploy new TiDB clusters that enable this feature on other Kubernetes clusters and join the initial TiDB cluster.

Experimentally supported scenarios:

- Enable this feature for a cluster that already has data. If you need to perform this action in a production environment, it is recommended to complete this requirement through data migration.

Unsupported scenarios:

- You cannot interconnect two clusters that already have data. You might perform this action through data migration.

4.6.3.3 Deploy a cluster across multiple Kubernetes clusters

Before you deploy a TiDB cluster across multiple Kubernetes clusters, you need to first deploy the Kubernetes clusters required for this operation. The following deployment assumes that you have completed Kubernetes deployment.

The following takes the deployment of one TiDB cluster across two Kubernetes clusters as an example. One TidbCluster is deployed in each Kubernetes cluster.

In the following sections, `${tc_name_1}` and `${tc_name_2}` refer to the name of TidbCluster that will be deployed in each Kubernetes cluster. `${namespace_1}` and `$ ↪ {namespace_2}` refer to the namespace of TidbCluster. `${cluster_domain_1}` and `${cluster_domain_2}` refer to the Cluster Domain of each Kubernetes cluster.

4.6.3.3.1 Step 1. Deploy the initial TidbCluster

Create and deploy the initial TidbCluster.

```
cat << EOF | kubectl apply -n ${namespace_1} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: "${tc_name_1}"
spec:
  version: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  enableDynamicConfiguration: true
  configUpdateStrategy: RollingUpdate
  clusterDomain: "${cluster_domain_1}"
  acrossK8s: true
  discovery: {}
  pd:
    baseImage: pingcap/pd
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    replicas: 1
    service:
      type: ClusterIP
    config: {}
EOF
```

The descriptions of the related fields are as follows:

- `spec.acrossK8s`: Specifies whether the TiDB cluster is deployed across Kubernetes clusters. In this example, this field must be set to `true`.

- `spec.clusterDomain`: If this field is set, the Pod FQDN which contains the cluster domain is used as the address for inter-component access.

  Take Pod `${tc_name}-pd-0` as an example: Pods in other Kubernetes clusters access this Pod using the `${tc_name}-pd-0.${tc_name}-pd-peer.${ns}.svc.${cluster_domain}` address.

  If the cluster domain is required when Pods access the Pod FQDB of another Kubernetes cluster, you must set this field.

4.6.3.3.2 Step 2. Deploy the new TidbCluster to join the TiDB cluster

After the initial cluster completes the deployment, you can deploy the new TidbCluster to join the TiDB cluster. You can create a new TidbCluster to join any existing TidbCluster.

```
cat << EOF | kubectl apply -n ${namespace_2} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: "${tc_name_2}"
spec:
  version: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  enableDynamicConfiguration: true
  configUpdateStrategy: RollingUpdate
  clusterDomain: "${cluster_domain_2}"
  acrossK8s: true
  cluster:
    name: "${tc_name_1}"
    namespace: "${namespace_1}"
    clusterDomain: "${cluster_domain_1}"
  discovery: {}
  pd:
    baseImage: pingcap/pd
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config: {}
```

```
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    replicas: 1
    service:
      type: ClusterIP
    config: {}
EOF
```

4.6.3.4 Deploy the TLS-enabled TiDB cluster across multiple Kubernetes clusters

You can follow the steps below to enable TLS between TiDB components for TiDB clusters deployed across multiple Kubernetes clusters.

The following takes the deployment of a TiDB cluster across two Kubernetes clusters as an example. One TidbCluster is deployed in each Kubernetes cluster.

In the following sections, `${tc_name_1}` and `${tc_name_2}` refer to the name of TidbCluster that will be deployed in each Kubernetes cluster. `${namespace_1}` and `${namespace_2}` refer to the namespace of TidbCluster. `${cluster_domain_1}` and `${cluster_domain_2}` refer to the Cluster Domain of each Kubernetes cluster.

4.6.3.4.1 Step 1. Issue the root certificate

Use `cfssl`

If you use `cfssl`, the CA certificate issue process is the same as the general issue process. You need to save the CA certificate created for the first time, and use this CA certificate when you issue certificates for TiDB components later.

In other words, when you create a component certificate in a cluster, you do not need to create a CA certificate again. Complete step 1 ~ 4 in Enabling TLS between TiDB components once to issue the CA certificate. After that, start from step 5 to issue certificates between other cluster components.

Use `cert-manager`

If you use `cert-manager`, you only need to create a `CA Issuer` and a `CA Certificate` in the initial cluster, and export the `CA Secret` to other new clusters that want to join.

For other clusters, you only need to create a component certificate `Issuer` (refers to `${cluster_name}-tidb-issuer` in the TLS document) and configure the `Issuer` to use the `CA`. The detailed process is as follows:

1. Create a `CA Issuer` and a `CA Certificate` in the initial Kubernetes cluster.

   Run the following command:

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ${tc_name_1}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tc_name_1}-ca
  namespace: ${namespace_1}
spec:
  secretName: ${tc_name_1}-ca-secret
  commonName: "TiDB"
  isCA: true
  duration: 87600h # 10yrs
  renewBefore: 720h # 30d
  issuerRef:
    name: ${tc_name_1}-selfsigned-ca-issuer
    kind: Issuer
EOF
```

2. Export the CA and delete irrelevant information.

   First, you need to export the `Secret` that stores the CA. The name of the `Secret` can be obtained from `.spec.secretName` of the `Certificate` YAML file in the first step.

```
kubectl get secret ${tc_name_1}-ca-secret -n ${namespace_1} -o yaml >
    ↪ ca.yaml
```

   Delete irrelevant information in the Secret YAML file. After the deletion, the YAML file is as follows (the information in `data` is omitted):

```
apiVersion: v1
data:
  ca.crt: LS0...LQo=
  tls.crt: LS0t....LQo=
  tls.key: LS0t...tCg==
kind: Secret
metadata:
  name: ${tc_name_2}-ca-secret
type: kubernetes.io/tls
```

3. Import the exported CA to other clusters.

   You need to configure the `namespace` so that related components can access the CA certificate:

```
kubectl apply -f ca.yaml -n ${namespace_2}
```

4. Create a component certificate `Issuer` in all Kubernetes clusters and configure it to use this CA.

   1. In the initial Kubernetes cluster, create an `Issuer` that issues certificates between TiDB components.

      Run the following command:

```
cat << EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ${tc_name_1}-tidb-issuer
  namespace: ${namespace_1}
spec:
  ca:
    secretName: ${tc_name_1}-ca-secret
EOF
```

   2. In other Kubernetes clusters, create an `Issuer` that issues certificates between TiDB components.

      Run the following command:

```
bash   cat << EOF | kubectl apply -f - apiVersion: cert-manager.io/v1
↪ kind: Issuer metadata:  name: ${tc_name_2}-tidb-issuer namespace: $
↪ {namespace_2} spec:    ca:       secretName: ${tc_name_2}-ca-secret
↪ EOF
```

4.6.3.4.2 Step 2. Issue certificates for the TiDB components of each Kubernetes cluster

You need to issue a component certificate for each TiDB component on the Kubernetes cluster. When issuing a component certificate, you need to add an authorization record ending with `.${cluster_domain}` to the hosts, for example, the record of the initial TidbCluster is `${tc_name_1}-pd.${namespace_1}.svc.${cluster_domain_1}`.

Use the `cfssl` system to issue certificates for TiDB components

The following example shows how to use `cfssl` to create a certificate used by PD. Run the following command to create the `pd-server.json` file for the initial TidbCluster.

```
cat << EOF > pd-server.json
{
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${tc_name_1}-pd",
      "${tc_name_1}-pd.${namespace_1}",
      "${tc_name_1}-pd.${namespace_1}.svc",
      "${tc_name_1}-pd.${namespace_1}.svc.${cluster_domain_1}",
      "${tc_name_1}-pd-peer",
      "${tc_name_1}-pd-peer.${namespace_1}",
      "${tc_name_1}-pd-peer.${namespace_1}.svc",
      "${tc_name_1}-pd-peer.${namespace_1}.svc.${cluster_domain_1}",
      "*.${tc_name_1}-pd-peer",
      "*.${tc_name_1}-pd-peer.${namespace_1}",
      "*.${tc_name_1}-pd-peer.${namespace_1}.svc",
      "*.${tc_name_1}-pd-peer.${namespace_1}.svc.${cluster_domain_1}"
    ],
    "key": {
        "algo": "ecdsa",
        "size": 256
    },
    "names": [
        {
            "C": "US",
            "L": "CA",
            "ST": "San Francisco"
        }
    ]
}
EOF
```

Use the `cert-manager` system to issue certificates for TiDB components

The following example shows how to use `cert-manager` to create a certificate used by PD for the initial TidbCluster. `Certificates` is shown below.

```
cat << EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tc_name_1}-pd-cluster-secret
  namespace: ${namespace_1}
spec:
```

```
  secretName: ${tc_name_1}-pd-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
    - "${tc_name_1}-pd"
    - "${tc_name_1}-pd.${namespace_1}"
    - "${tc_name_1}-pd.${namespace_1}.svc"
    - "${tc_name_1}-pd.${namespace_1}.svc.${cluster_domain_1}"
    - "${tc_name_1}-pd-peer"
    - "${tc_name_1}-pd-peer.${namespace_1}"
    - "${tc_name_1}-pd-peer.${namespace_1}.svc"
    - "${tc_name_1}-pd-peer.${namespace_1}.svc.${cluster_domain_1}"
    - "*.${tc_name_1}-pd-peer"
    - "*.${tc_name_1}-pd-peer.${namespace_1}"
    - "*.${tc_name_1}-pd-peer.${namespace_1}.svc"
    - "*.${tc_name_1}-pd-peer.${namespace_1}.svc.${cluster_domain_1}"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${tc_name_1}-tidb-issuer
    kind: Issuer
    group: cert-manager.io
EOF
```

You need to refer to the TLS-related documents, issue the corresponding certificates for the components, and create the `Secret` in the corresponding Kubernetes clusters.

For other TLS-related information, refer to the following documents:

- Enable TLS between TiDB Components
- Enable TLS for the MySQL Client

4.6.3.4.3 Step 3. Deploy the initial TidbCluster

Run the following commands to deploy the initial TidbCluster. The following `YAML` file enables the TLS feature and configures `cert-allowed-cn`, which makes each component start to verify the certificates issued by the `CN` for the `CA` of `TiDB`.

```
cat << EOF | kubectl apply -n ${namespace_1} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: "${tc_name_1}"
spec:
  version: v8.5.0
  timezone: UTC
  tlsCluster:
   enabled: true
  pvReclaimPolicy: Delete
  enableDynamicConfiguration: true
  configUpdateStrategy: RollingUpdate
  clusterDomain: "${cluster_domain_1}"
  acrossK8s: true
  discovery: {}
  pd:
    baseImage: pingcap/pd
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config:
      security:
        cert-allowed-cn:
          - TiDB
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config:
      security:
       cert-allowed-cn:
         - TiDB
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    replicas: 1
    service:
      type: ClusterIP
    tlsClient:
      enabled: true
```

```
    config:
      security:
       cert-allowed-cn:
         - TiDB
EOF
```

4.6.3.4.4 Step 4. Deploy a new TidbCluster to join the TiDB cluster

After the initial cluster completes the deployment, you can deploy the new TidbCluster to join the TiDB cluster. You can create a new TidbCluster to join any existing TidbCluster.

```
cat << EOF | kubectl apply -n ${namespace_2} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: "${tc_name_2}"
spec:
  version: v8.5.0
  timezone: UTC
  tlsCluster:
   enabled: true
  pvReclaimPolicy: Delete
  enableDynamicConfiguration: true
  configUpdateStrategy: RollingUpdate
  clusterDomain: "${cluster_domain_2}"
  acrossK8s: true
  cluster:
    name: "${tc_name_1}"
    namespace: "${namespace_1}"
    clusterDomain: "${cluster_domain_1}"
  discovery: {}
  pd:
    baseImage: pingcap/pd
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config:
      security:
        cert-allowed-cn:
          - TiDB
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 1
```

```
    requests:
      storage: "10Gi"
    config:
      security:
       cert-allowed-cn:
         - TiDB
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    replicas: 1
    service:
      type: ClusterIP
    tlsClient:
      enabled: true
    config:
      security:
       cert-allowed-cn:
         - TiDB
EOF
```

4.6.3.5 Upgrade TiDB Cluster

For a TiDB cluster deployed across Kubernetes clusters, to perform a rolling upgrade for each component Pod of the TiDB cluster, take the following steps in sequence to modify the `version` configuration of each component in the TidbCluster spec for each Kubernetes cluster.

1. Upgrade PD versions for all Kubernetes clusters.
2. Modify the `spec.pd.version` field in the spec for the initial TidbCluster.

   ```
   yaml  apiVersion: pingcap.com/v1alpha1 kind: TidbCluster # ... spec:
   ↪ pd:      version: ${version}
   ```

   2. Watch the status of PD Pods and wait for PD Pods in the initial TidbCluster to finish recreation and become `Running`.
   3. Repeat the first two substeps to upgrade all PD Pods in other TidbCluster.

3. Take step 1 as an example, perform the following upgrade operations in sequence:

   1. If PD microservices (introduced in TiDB v8.0.0) are deployed in clusters, upgrade the version of PD microservices for all Kubernetes clusters that have PD microservices deployed.
   2. If TiProxy is deployed in clusters, upgrade the TiProxy versions for all the Kubernetes clusters that have TiProxy deployed.

3. If TiFlash is deployed in clusters, upgrade the TiFlash versions for all the Kubernetes clusters that have TiFlash deployed.
4. Upgrade TiKV versions for all Kubernetes clusters.
5. If Pump is deployed in clusters, upgrade the Pump versions for all the Kubernetes clusters that have Pump deployed.
6. Upgrade TiDB versions for all Kubernetes clusters.
7. If TiCDC is deployed in clusters, upgrade the TiCDC versions for all the Kubernetes clusters that have TiCDC deployed.

4.6.3.6 Exit and reclaim TidbCluster that already join a cross-Kubernetes cluster

When you need to make a cluster exit from the joined TiDB cluster deployed across Kubernetes and reclaim resources, you can perform the operation by scaling in the cluster. In this scenario, the following requirements of scaling-in need to be met.

- After scaling in the cluster, the number of TiKV replicas in the cluster should be greater than the number of `max-replicas` set in PD. By default, the number of TiKV replicas needs to be greater than three.

Take the second TidbCluster created in the last section as an example. First, set the number of replicas of PD, TiKV, and TiDB to `0`. If you have enabled other components such as TiFlash, TiCDC, TiProxy, and Pump, set the number of these replicas to `0`:

> **Note:**
>
> Starting from v8.0.0, PD supports the microservice mode. If PD microservices are configured, you also need to set the `replicas` of the corresponding PD microservice component to `0` in the `pdms` configuration.

```
kubectl patch tc ${tc_name_2} -n ${namespace_2} --type merge -p '{"spec":{"
    ↪ pd":{"replicas":0},"tikv":{"replicas":0},"tidb":{"replicas":0}}}'
```

Wait for the status of the second TidbCluster to become `Ready`, and scale in related components to `0` replica:

```
kubectl get pods -l app.kubernetes.io/instance=${tc_name_2} -n ${namespace_2
    ↪ }
```

The Pod list shows `No resources found`. At this time, all Pods have been scaled in, and the second TidbCluster exits the cluster. Check the cluster status of the second TidbCluster:

```
kubectl get tc ${tc_name_2} -n ${namespace_2}
```

The result shows that the second TidbCluster is in the `Ready` status. At this time, you can delete the object and reclaim related resources.

```
kubectl delete tc ${tc_name_2} -n ${namespace_2}
```

Through the above steps, you can complete exit and resources reclaim of the joined clusters.

4.6.3.7 Enable the feature for a cluster with existing data and make it the initial TiDB cluster

> **Warning:**
>
> Currently, this is an experimental feature and might cause data loss. Please use it carefully.

A cluster with existing data refer to a deployed TiDB cluster with the configuration `spec.acrossK8s: false`.

Depending on the network between multiple Kubernetes clusters, there are different methods.

If all Kubernetes have the same Cluster Domain, you only need to update the `spec.crossK8s` configuration of TidbCluster. Run the following command:

```
kubectl patch tidbcluster cluster1 --type merge -p '{"spec":{"acrossK8s":
    ↪ true}}'
```

After the modification, wait for the TiDB cluster to complete rolling update.

If each Kubernetes have different Cluster Domain, you need to update the `spec.clusterDomain` and `spec.acrossK8s` fields. Take the following steps:

1. Update the `spec.clusterDomain` and `spec.acrossK8s` fields:

   Configure the following parameters according to the `clusterDomain` in your Kubernetes cluster information:

   > **Warning:**
   >
   > Currently, you need to configure `clusterDomain` with correct information. After modifying the configuration, you can not modify it again.

```
kubectl patch tidbcluster cluster1 --type merge -p '{"spec":{"
    ↪ clusterDomain":"cluster1.com", "acrossK8s": true}}'
```

After completing the modification, the TiDB cluster performs the rolling update.

2. Update the `PeerURL` information of PD:

   After completing the rolling update, you need to use `port-forward` to expose PD's API, and use API of PD to update `PeerURL` of PD.

   1. Use `port-forward` to expose API of PD:

      ```
      kubectl port-forward pods/cluster1-pd-0 2380:2380 2379:2379 -n
          ↪ pingcap
      ```

   2. Access `PD API` to obtain `members` information. Note that after using `port-forward`, the terminal session is occupied. You need to perform the following operations in another terminal session:

      ```
      curl http://127.0.0.1:2379/v2/members
      ```

> **Note:**
> If the cluster enables TLS, you need to configure the certificate when using the curl command. For example:
> `curl --cacert /var/lib/pd-tls/ca.crt --cert /var/lib/pd-tls/tls.crt --key /var/lib/pd-tls/tls.key https://127.0.0.1:2379/v2/members`

After running the command, the output is as follows:

```
{"members":[{"id":"6ed0312dc663b885","name":"cluster1-pd-0.cluster1
    ↪ -pd-peer.pingcap.svc.cluster1.com","peerURLs":["http://
    ↪ cluster1-pd-0.cluster1-pd-peer.pingcap.svc:2380"],"
    ↪ clientURLs":["http://cluster1-pd-0.cluster1-pd-peer.pingcap.
    ↪ svc.cluster1.com:2379"]},{"id":"bd9acd3d57e24a32","name":"
    ↪ cluster1-pd-1.cluster1-pd-peer.pingcap.svc.cluster1.com","
    ↪ peerURLs":["http://cluster1-pd-1.cluster1-pd-peer.pingcap.
    ↪ svc:2380"],"clientURLs":["http://cluster1-pd-1.cluster1-pd-
    ↪ peer.pingcap.svc.cluster1.com:2379"]},{"id":"
    ↪ e04e42cccef60246","name":"cluster1-pd-2.cluster1-pd-peer.
    ↪ pingcap.svc.cluster1.com","peerURLs":["http://cluster1-pd-2.
    ↪ cluster1-pd-peer.pingcap.svc:2380"],"clientURLs":["http://
    ↪ cluster1-pd-2.cluster1-pd-peer.pingcap.svc.cluster1.com
    ↪ :2379"]}]}
```

3. Record the `id` of each PD instance, and use the `id` to update the `peerURL` of each member in turn:

```
member_ID="6ed0312dc663b885"
member_peer_url="http://cluster1-pd-0.cluster1-pd-peer.pingcap.svc.
    ↪ cluster1.com:2380"
curl http://127.0.0.1:2379/v2/members/${member_ID} -XPUT \
-H "Content-Type: application/json" -d '{"peerURLs":["${
    ↪ member_peer_url}"]}'
```

After completing the above steps, this TidbCluster can be used as the initial TidbCluster for TiDB cluster deployment across Kubernetes clusters. You can refer the section to deploy other TidbCluster.

For more examples and development information, refer to `multi-cluster`.

4.6.3.8 Deploy TiDB monitoring components

Refer to Deploy TiDB Monitor across Multiple Kubernetes Clusters.

4.7 Deploy a Heterogeneous Cluster for an Existing TiDB Cluster

This document describes how to deploy a heterogeneous cluster for an existing TiDB cluster. A heterogeneous cluster consists of nodes with different configurations from the existing TiDB cluster.

4.7.1 Usage scenarios

This document is applicable to scenarios in which you need to create differentiated instances for an existing TiDB cluster, such as the following:

- Create TiKV clusters with different configurations and different labels for hotspot scheduling.
- Create TiDB clusters with different configurations for OLTP and OLAP queries.

4.7.2 Prerequisites

You already have a TiDB cluster. If not, deploy a TiDB cluster on Kubernetes first.

4.7.3 Deploy a heterogeneous cluster

Depending on whether you need to enable Transport Layer Security (TLS) for a heterogeneous cluster, choose one of the following methods:

- Deploy a heterogeneous cluster
- Deploy a TLS-enabled heterogeneous cluster

To deploy a heterogeneous cluster, do the following:

1. Create a cluster configuration file for the heterogeneous cluster.

   Run the following command to create a cluster configuration file for the heterogeneous cluster. Replace `${origin_cluster_name}` with the name of the existing cluster, and replace `${heterogeneous_cluster_name}` with the name of the heterogeneous cluster. To view the monitoring data of both the existing cluster and the heterogeneous cluster in the same Grafana of TidbMonitor, you need to name the heterogeneous cluster with the prefix of the existing cluster name.

> **Note:**
>
> Comparing with the configuration file of a normal TiDB cluster, the only difference in the configuration file of a heterogeneous TiDB cluster is that you need to additionally specify the `spec.cluster.name` field as the name of an existing TiDB cluster. According to this field, TiDB Operator adds the heterogeneous cluster to the existing TiDB cluster.

```
origin_cluster_name=basic
heterogeneous_cluster_name=basic-heterog
cat > cluster.yaml << EOF
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${heterogeneous_cluster_name}
spec:
  configUpdateStrategy: RollingUpdate
  version: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  discovery: {}
  cluster:
    name: ${origin_cluster_name}
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 1
    # If storageClassName is not set, the default Storage Class of the
      ↪ Kubernetes cluster is used.
    # storageClassName: local-storage
```

```
    requests:
      storage: "100Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    replicas: 1
    service:
      type: ClusterIP
    config: {}
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 0
    replicas: 1
    storageClaims:
      - resources:
          requests:
            storage: 100Gi
EOF
```

For more configurations and field meanings of TiDB cluster, see the TiDB cluster configuration document.

2. In the configuration file of your heterogeneous cluster, modify the configurations of each node according to your need.

   For example, you can modify the number of `replicas` for each component in the `cluster.yaml` file, or remove components that are not needed.

3. Create the heterogeneous cluster by running the following command. You need to replace `cluster.yaml` with the configuration filename of your heterogeneous cluster.

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

   If the output shows `tidbcluster.pingcap.com/${heterogeneous_cluster_name}` ↪ `created`, the execution is successful. Then, TiDB Operator will create the TiDB cluster with the configurations according to the cluster configuration file.

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

After creating certificates, take the following steps to deploy a TLS-enabled heterogeneous cluster.

1. Create a cluster configuration file for the heterogeneous cluster.

   For example, 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.

> **Note:**
> Comparing with the configuration file of a normal TiDB cluster, the only difference in the configuration file of a heterogeneous TiDB cluster is that you need to additionally specify the `spec.cluster.name` field as the name of an existing TiDB cluster. According to this field, TiDB Operator adds the heterogeneous cluster to the existing TiDB cluster.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${heterogeneous_cluster_name}
spec:
  tlsCluster:
    enabled: true
  configUpdateStrategy: RollingUpdate
  version: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  discovery: {}
  cluster:
    name: ${origin_cluster_name}
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 1
    # If storageClassName is not set, the default Storage Class of the
       ↪ Kubernetes cluster is used.
    # storageClassName: local-storage
    requests:
      storage: "100Gi"
    config: {}
```

```
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    replicas: 1
    service:
      type: ClusterIP
    config: {}
    tlsClient:
      enabled: true
  tiflash:
    baseImage: pingcap/tiflash
    maxFailoverCount: 0
    replicas: 1
    storageClaims:
      - resources:
          requests:
            storage: 100Gi
```

In the configuration file, `spec.tlsCluster.enabled`controls whether to enable TLS between the components and `spec.tidb.tlsClient.enabled`controls whether to enable TLS for the MySQL client.

- For more configurations of a TLS-enabled heterogeneous cluster, see the 'heterogeneous-tls' example.
- For more configurations and field meanings of a TiDB cluster, see the TiDB cluster configuration document.

2. In the configuration file of your heterogeneous cluster, modify the configurations of each node according to your need.

   For example, you can modify the number of `replicas` for each component in the `cluster.yaml` file, or remove components that are not needed.

3. Create the TLS-enabled heterogeneous cluster by running the following command. You need to replace `cluster.yaml` with the configuration filename of the heterogeneous cluster.

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

   If the output shows `tidbcluster.pingcap.com/${heterogeneous_cluster_name}` ↪ `created`, the execution is successful. Then, TiDB Operator will create the TiDB cluster with the configurations according to your cluster configuration file.

4.7.3.1 Deploy a cluster monitoring component

If you need to deploy a monitoring component for a heterogeneous cluster, take the following steps to add the heterogeneous cluster name to the TidbMonitor CR file of an existing TiDB cluster.

1. Edit the TidbMonitor Custom Resource (CR) of the existing TiDB cluster:

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

2. Replace `${heterogeneous_cluster_name}` with the desired name of your heterogeneous cluster, and replace `${origin_cluster_name}` with the name of the existing cluster. For example:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
name: heterogeneous
spec:
clusters:
    - name: ${origin_cluster_name}
    - name: ${heterogeneous_cluster_name}
prometheus:
    baseImage: prom/prometheus
    version: v2.27.1
grafana:
    baseImage: grafana/grafana
    version: 7.5.11
initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v8.5.0
reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
imagePullPolicy: IfNotPresent
```

4.8 Deploy TiCDC on Kubernetes

TiCDC is a tool for replicating the incremental data of TiDB. This document describes how to deploy TiCDC on Kubernetes using TiDB Operator.

You can deploy TiCDC when deploying a new TiDB cluster, or add the TiCDC component to an existing TiDB cluster.

4.8.1 Prerequisites

TiDB Operator is deployed.

4.8.2 Fresh TiCDC deployment

To deploy TiCDC when deploying the TiDB cluster, refer to Deploy TiDB on General Kubernetes.

4.8.3 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. Mount persistent volumes (PVs) for TiCDC。

   TiCDC supports mounting multiple PV. It is recommended that you plan the number of PVs required when deploying TiCDC for the first time. For more information, refer to Multiple disks mounting。

4. After the deployment, enter a TiCDC Pod by running `kubectl exec`:

```
kubectl exec -it ${pod_name} -n ${namespace} -- sh
```

5. Manage the cluster and data replication tasks by using `cdc cli`.

```
# The default port for TiCDC server deployed through TiDB operator is
    ↪ 8301
/cdc cli capture list --server=http://127.0.0.1:8301
```

```
[
  {
    "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 --server=http://127.0.0.1:8301 --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.

4.9 Deploy TiDB Binlog

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

> **Warning:**
>
> Starting from TiDB v7.5.0, TiDB Binlog replication is deprecated. Starting from v8.3.0, TiDB Binlog is fully deprecated, with removal planned for a future release. For incremental data replication, use TiCDC instead. For point-in-time recovery (PITR), use PITR.

4.9.1 Prerequisites

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

4.9.2 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.

4.9.2.1 Deploy Pump

1. Modify the `TidbCluster` CR file to add the Pump configuration.

   For example:

```
spec:
  ...
  pump:
    baseImage: pingcap/tidb-binlog
    version: v8.1.0
    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: v8.1.0
    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.

4.9.3 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: v8.1.0
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 on 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.

4.9.4 Enable TLS

4.9.4.1 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
...
```

4.9.4.2 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
...
```

4.9.5 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
```

4.9.5.1 Scale in Pump nodes

1. Scale in Pump Pods:

   ```
   kubectl patch tc ${cluster_name} -n ${namespace} --type merge -p '{"
       ↪ spec":{"pump":{"replicas": ${pump_replicas}}}}'
   ```

   In the command above, `${pump_replicas}` is the desired number of Pump Pods after the scaling.

   > **Note:**
   > Do not scale in Pump nodes to 0. Otherwise, Pump nodes are removed completely.

2. Wait for the Pump Pods to automatically be taken offline and deleted. Run the following command to observe the Pod status:

   ```
   watch kubectl get po ${cluster_name} -n ${namespace}
   ```

3. (Optional) Force Pump to go offline:

   If the offline operation fails, that is, the Pump Pods are not deleted for a long time, you can forcibly mark Pump as `offline`.

- 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"
```

4.9.5.2 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 patch tc ${tidb-
  ↪ cluster} -n ${namespace} --type merge -p '{"spec":{"pump
  ↪ ":{"replicas": 1}}}'`.

1. Before removing Pump nodes, execute `kubectl patch tc ${cluster_name} -n ${
   ↪ namespace} --type merge -p '{"spec":{"tidb":{"binlogEnabled": 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 patch tc ${cluster_name} -n ${namespace} --type json -p
   ↪ '[{"op":"remove", "path":"/spec/pump"}]'` to 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}`.

4.9.5.3 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"
   ```

5 Monitor and Alert

5.1 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.

5.1.1 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:

- `spec.clusters[].name` should be set to the `TidbCluster` name of the corresponding TiDB cluster.

5.1.1.1 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: v2.27.1
    service:
      type: NodePort
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
    service:
      type: NodePort
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v8.5.0
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
  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
```

5.1.1.2 Customize the Prometheus configuration

You can customize the Prometheus configuration by using a customized configuration file or by adding extra options to the command.

5.1.1.2.1 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.
3. Check if TidbMonitor has enabled dynamic configuration. If not, you need to restart TidbMonitor's pod to reload the configuration.

For the complete configuration, refer to the tidb-operator example.

5.1.1.2.2 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

5.1.1.3 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.

5.1.1.4 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`.

5.1.1.5 Set kube-prometheus and AlertManager

Nodes-Info and Pods-Info monitoring dashboards are built into TidbMonitor Grafana by default to view the corresponding monitoring metrics of Kubernetes.

To view these monitoring metrics in TidbMonitor Grafana, take the following steps:

1. Deploy Kubernetes cluster monitoring manually.

There are multiple ways to deploy Kubernetes cluster monitoring. To use kube-prometheus for deployment, see the kube-prometheus documentation.

2. Set the `TidbMonitor.spec.kubePrometheusURL` to obtain Kubernetes monitoring data.

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: v2.27.1
    service:
      type: NodePort
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
    service:
      type: NodePort
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v8.5.0
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
  imagePullPolicy: IfNotPresent
```

5.1.2 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.

5.1.2.1 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.

5.1.2.2 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: v2.27.1
    ingress:
      hosts:
      - example.com
      annotations:
        foo: "bar"
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
    service:
      type: ClusterIP
    ingress:
      hosts:
        - example.com
      annotations:
        foo: "bar"
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v8.5.0
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
```

```
  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: v2.27.1
    ingress:
      hosts:
      - example.com
      tls:
      - hosts:
        - example.com
        secretName: testsecret-tls
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
    service:
      type: ClusterIP
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v8.5.0
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
  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}
```

5.1.3 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.

5.1.4 Monitor multiple clusters

Starting from TiDB Operator 1.2, TidbMonitor supports monitoring multiple clusters across namespaces.

5.1.4.1 Configure the monitoring of multiple clusters using YAML files

For the clusters to be monitored, regardless of whether `TLS` is enabled or not, you can monitor them by configuring TidbMonitor's YAML file.

A configuration example is as follows:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: basic
spec:
  clusterScoped: true
  clusters:
    - name: ns1
      namespace: ns1
    - name: ns2
      namespace: ns2
  persistent: true
  storage: 5G
  prometheus:
    baseImage: prom/prometheus
    version: v2.27.1
    service:
      type: NodePort
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
    service:
      type: NodePort
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v8.5.0
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
  imagePullPolicy: IfNotPresent
```

For a complete configuration example, refer to Example in the TiDB Operator repository.

5.1.4.2 Monitor multiple clusters using Grafana

If the `tidb-monitor-initializer` image is earlier than v4.0.14 or v5.0.3, to monitor multiple clusters, you can take the following steps in each Grafana Dashboard:

1. On Grafana Dashboard, click **Dashboard settings** to open the **Settings** panel.
2. On the **Settings** panel, select the **tidb_cluster** variable from **Variables**, and then set the **Hide** property of the **tidb_cluster** variable to the null option in the drop-down list.
3. Get back to the current Grafana Dashboard (changes to the **Hide** property cannot be saved currently), and you can see the drop-down list for cluster selection. The cluster name in the drop-down list is in the `${namespace}-${name}` format.

If you need to save changes to the Grafana Dashboard, Grafana must be `6.5` or later, and TiDB Operator must be v1.2.0-rc.2 or later.

5.2 Access TiDB Dashboard

TiDB Dashboard is a visualized tool introduced since TiDB v4.0 and is used to monitor and diagnose TiDB clusters. For details, see TiDB Dashboard.

This document describes how to access TiDB Dashboard on Kubernetes.

- In a test environment, you can access TiDB Dashboard by port forward.
- In a production environment, it is recommended to access TiDB Dashboard by Ingress. You can also enable the TLS transfer. See Use Ingress with TLS for details.
- To access TiDB Dashboard without a domain name, you can use NodePort Service.

> **Note:**
>
> Due to the special environment of Kubernetes, some features of TiDB Dashboard are not supported in TiDB Operator. See Unsupported TiDB Dashboard features for details.

In this document, you can use the Discovery service to access TiDB Dashboard. 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.

5.2.1 Prerequisites: Determine the TiDB Dashboard service

This section describes how to determine the TiDB Dashboard `service` and `HTTP` path in different deployment methods of TiDB Dashboard. You can fill in the `service` and `HTTP` path obtained in this section in the target configuration file to access TiDB Dashboard.

TiDB supports two methods to deploy TiDB Dashboard. You can choose one of the two methods to access TiDB Dashboard:

- Deployed as an independent service. In this deployment method, TiDB Dashboard is an independent StatefulSet and has a dedicated service. The Web server path can be configured through `TidbDashboard.spec.pathPrefix`.
- Built in PD. The TiDB Dashboard deployed in this method is available in the `/dashboard` path of the PD. Other paths outside of this might not have access control. Note that this deployment method will be removed in future TiDB releases. Therefore, it is recommended to deploy TiDB Dashboard as an independent service.

5.2.1.1 Access TiDB Dashboard built in PD

To access TiDB Dashboard built in PD, 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
```

In this deployment method, the `service`, `port`, and `HTTP` paths of TiDB Dashboard are as follows:

```
export SERVICE_NAME=${cluster_name}-discovery && \
export PORT=10261 && \
export HTTP_PATH=/dashboard
```

5.2.1.2 Access independently deployed TiDB Dashboard

To access an independently deployed TiDB Dashboard, you need to use TiDB Operator v1.4.0 (or later versions) and the TiDB cluster v4.0.1 (or later versions).

Before accessing TiDB Dashboard, ensure that you have deployed an independent TiDB Dashboard.

In this deployment method, the `service`, `port`, and `HTTP` paths of TiDB Dashboard are as follows (default values):

```
export SERVICE_NAME=${cluster_name}-tidb-dashboard-exposed && \
export PORT=12333 && \
export HTTP_PATH=""
```

5.2.2 Method 1. 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 TiDB cluster on Kubernetes.

```
kubectl port-forward svc/${SERVICE_NAME} -n ${namespace} ${PORT}:${PORT}
```

In the preceding command:

- `${namespace}` is `TidbCluster.namespace`.
- `port-forward` binds to the IP address 127.0.0.1 by default. 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.

Visit `http://localhost:${PORT}${HTTP_PATH}` in your browser to access TiDB Dashboard.

5.2.3 Method 2. Access TiDB Dashboard by Ingress

In important production environments, it is recommended to expose the TiDB Dashboard service using Ingress.

5.2.3.1 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.

5.2.3.2 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: ${SERVICE_NAME}
              servicePort: ${PORT}
            path: ${HTTP_PATH}
```

2. After Ingress is deployed, you can access TiDB Dashboard via `http://${host}${path}` outside the Kubernetes cluster.

5.2.3.3 Use Ingress with 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: ${SERVICE_NAME}
            servicePort: ${PORT}
          path: ${HTTP_PATH}
```

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://${host}${path}` to access TiDB Dashboard.

5.2.4 Method 3. 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.

5.2.4.1 Access TiDB Dashboard built in PD

To access TiDB Dashboard built in PD, you need to create a `NodePort` service for PD.

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.

5.2.4.2 Access independently deployed TiDB Dashboard

> **Note:**
>
> When deploying TiDB Dashboard independently, you need to set `TidbDashboard.spec.service.type` to `NodePort`.

After deploying TiDB Dashboard independently, you can get the `nodePort` of `${cluster_name}-tidb-dashboard-exposed` by running the `kubectl get svc` command, and then access TiDB Dashboard via `https://{nodeIP}:{nodePort}>`.

5.2.5 Enable Continuous Profiling

With Continuous Profiling, you can collect continuous performance data of TiDB, PD, TiKV, and TiFlash instances, and have the nodes monitored day and night without restarting any of them. The data collected can be displayed in various forms, for example, on a flame graph or a directed acyclic graph. The data displayed visually shows what internal operations are performed on the instances during the performance profiling period and the corresponding proportions. With such data, you can quickly learn the CPU resource consumption of these instances.

To enable this feature, you need to deploy TidbNGMonitoring CR using TiDB Operator v1.3.0 or later versions.

1. Deploy TidbMonitor CR.

    - If your TiDB cluster is earlier than v5.4.0, see Deploy Monitoring and Alerts for a TiDB Cluster to deploy TidbMonitor CR.

- If your TiDB cluster is v5.4.0 or later, skip this step.

2. Deploy TidbNGMonitoring CR.

   Run the following command to deploy TidbNGMonitoring CR. In this command, `${cluster_name}` is the name of the TidbCluster CR and `${cluster_ns}` is the namespace of this CR.

```
cat << EOF | kubectl apply -n ${ns} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbNGMonitoring
metadata:
  name: ${name}
spec:
  clusters:
  - name: ${cluster_name}
    namespace: ${cluster_ns}
  ngMonitoring:
    requests:
      storage: 10Gi
    version: v8.5.0
    # storageClassName: default
    baseImage: pingcap/ng-monitoring
EOF
```

   For more configuration items of the TidbNGMonitoring CR, see example in tidb-operator.

3. Enable Continuous Profiling.

   1. On TiDB Dashboard, click **Advanced Debugging** > **Profiling Instances** > **Continuous Profiling**.
   2. In the displayed window, click **Open Settings**. Switch on the button under **Enable Feature** on the right. Modify the value of **Retention Duration** as required or retain the default value.
   3. Click **Save** to enable this feature.

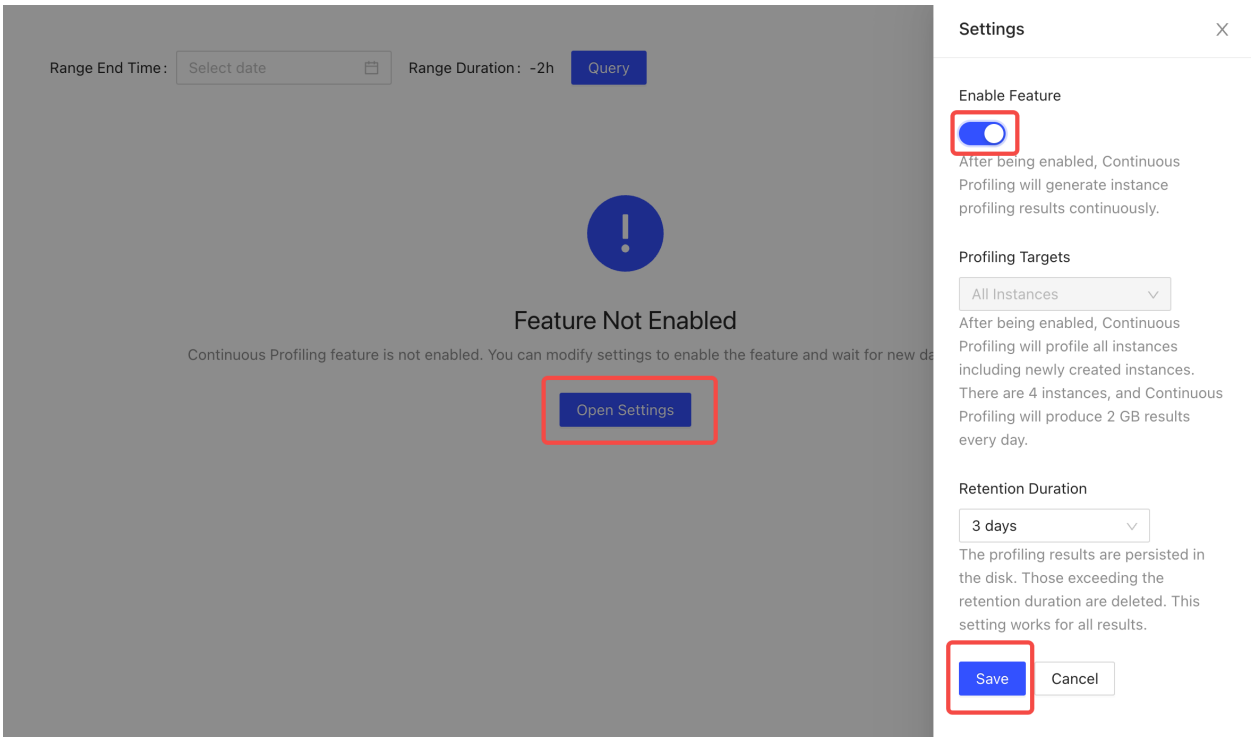


Figure 1: Enable the feature

For more operations of the Continuous Profiling function, see TiDB Dashboard Instance Profiling - Continuous Profiling.

5.2.6 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.

5.3 Aggregate Monitoring Data of Multiple TiDB Clusters

This document describes how to aggregate the monitoring data of multiple TiDB clusters by Thanos to provide centralized monitoring service.

5.3.1 Thanos

Thanos is a high availability solution for Prometheus that simplifies the availability guarantee of Prometheus.

Thanos provides Thanos Query component as a unified query solution across multiple Prometheus clusters. You can use this feature to aggregate monitoring data of multiple TiDB clusters.

5.3.2 Aggregate monitoring data via Thanos Query

5.3.2.1 Configure Thanos Query

1. Configure a Thanos Sidecar container for each TidbMonitor.

   A configuration example is as follows.

```
kubectl -n ${namespace} apply -f https://raw.githubusercontent.com/
    ↪ pingcap/tidb-operator/v1.6.1/examples/monitor-with-thanos/tidb-
    ↪ monitor.yaml
```

2. Deploy the Thanos Query component.

   1. Download the `thanos-query.yaml` file for Thanos Query deployment:

```
curl -sl -O https://raw.githubusercontent.com/pingcap/tidb-operator
    ↪ /v1.6.1/examples/monitor-with-thanos/thanos-query.yaml
```

   2. Manually modify the `--store` parameter in the `thanos-query.yaml` file by updating `basic-prometheus:10901` to `basic-prometheus.${namespace}:10901`. `${namespace}` is the namespace where TidbMonitor is deployed.
   3. Execute the `kubectl apply` command for deployment.

```
kubectl -n ${thanos_namespace} apply -f thanos-query.yaml
```

   In the command above, `${thanos_namespace}` is the namespace where the Thanos Query component is deployed.

In Thanos Query, a Prometheus instance corresponds to a store and also corresponds to a TidbMonitor. After deploying Thanos Query, you can provide a uniform query interface for monitoring data through Thanos Query's API.

5.3.2.2 Access the Thanos Query panel

To access the Thanos Query panel, execute the following command, and then access http://127.0.0.1:9090 in your browser:

```
kubectl port-forward -n ${thanos_namespace} svc/thanos-query 9090
```

If you want to access the Thanos Query panel using NodePort or LoadBalancer, refer to the following documents:

- NodePort method
- LoadBalancer way

5.3.2.3 Configure Grafana

After deploying Thanos Query, to query the monitoring data of multiple TidbMonitors, take the following steps:

1. Log in to Grafana.
2. In the left navigation bar, select `Configuration` > `Data Sources`.
3. Add or modify a DataSource in the Prometheus type.
4. Set the URL under HTTP to `http://thanos-query.${thanos_namespace}:9090`.

5.3.2.4 Add or remove TidbMonitor

In Thanos Query, a Prometheus instance corresponds to a monitor store and also corresponds to a TidbMonitor. If you need to add, update, or remove a monitor store from the Thanos Query, update the `--store` configuration of the Thanos Query component, and perform a rolling update to the Thanos Query component.

```
spec:
 containers:
   - args:
       - query
       - --grpc-address=0.0.0.0:10901
       - --http-address=0.0.0.0:9090
       - --log.level=debug
       - --log.format=logfmt
       - --query.replica-label=prometheus_replica
       - --query.replica-label=rule_replica
       - --store=<TidbMonitorName1>-prometheus.<TidbMonitorNs1>:10901
       - --store=<TidbMonitorName2>-prometheus.<TidbMonitorNs2>:10901
```

5.3.2.5 Configure archives and storage of Thanos Sidecar

> **Note:**
>
> To ensure successful configuration, you must first create the S3 bucket. If you choose AWS S3, refer to AWS documentation - Create AWS S3 Bucket and AWS documentation - AWS S3 Endpoint List for instructions.

Thanos Sidecar supports replicating monitoring data to S3 remote storage.

The configuration of the `TidbMonitor` CR is as follows:

```
spec:
  thanos:
    baseImage: thanosio/thanos
    version: v0.17.2
    objectStorageConfig:
      key: objectstorage.yaml
      name: thanos-objectstorage
```

Meanwhile, you need to create a Secret. The example is as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: thanos-objectstorage
type: Opaque
stringData:
  objectstorage.yaml: |
    type: S3
    config:
      bucket: "xxxxxx"
      endpoint: "xxxx"
      region: ""
      access_key: "xxxx"
      insecure: true
      signature_version2: true
      secret_key: "xxxx"
      put_user_metadata: {}
      http_config:
        idle_conn_timeout: 90s
        response_header_timeout: 2m
      trace:
        enable: true
```

```
    part_size: 41943040
```

5.3.3 RemoteWrite mode

Besides aggregating data via Thanos Query, you can also push monitoring data to Thanos using Prometheus' RemoteWrite feature.

To enable the RemoteWrite mode, specify the Prometheus RemoteWrite configuration when you create the TidbMonitor CR. For example:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: basic
spec:
  clusters:
  - name: basic
  prometheus:
    baseImage: prom/prometheus
    version: v2.27.1
    remoteWrite:
      - url: "http://thanos-receiver:19291/api/v1/receive"
  grafana:
    baseImage: grafana/grafana
    version: 7.5.11
  initializer:
    baseImage: registry.cn-beijing.aliyuncs.com/tidb/tidb-monitor-
       ↪ initializer
    version: v8.5.0
  reloader:
    baseImage: registry.cn-beijing.aliyuncs.com/tidb/tidb-monitor-reloader
    version: v1.0.1
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
  imagePullPolicy: IfNotPresent
```

After RemoteWrite is enabled, Prometheus pushes the monitoring data to Thanos Receiver. For more information, refer to the design of Thanos Receiver.

For details on the deployment, refer to this example of integrating TidbMonitor with Thanos Receiver.

5.4 Monitor a TiDB Cluster across Multiple Kubernetes Clusters

You can monitor a TiDB cluster that is deployed across multiple Kubernetes clusters and access the monitoring data from a global view. This document describes how to integrate with several popular multi-cluster monitoring solutions based on Prometheus, and how to use Grafana to visualize the data across multiple Kubernetes clusters:

5.4.1 Push data from Prometheus

The push method uses the remote write feature of Prometheus, which requests the Prometheus instances in different Kubernetes clusters to push monitoring data to a centralized storage.

The push method described in this section is based on Thanos. If you use other centralized storage solutions compatible with the Prometheus remote API, you only need to replace the related Thanos components.

5.4.1.1 Prerequisites

The multiple Kubernetes clusters must meet the following condition:

- The Prometheus (`TidbMonitor`) component in each Kubernetes cluster has access to the Thanos Receiver component.

For the deployment instructions of Thanos Receiver, refer to kube-thanos and the example.

5.4.1.2 Architecture

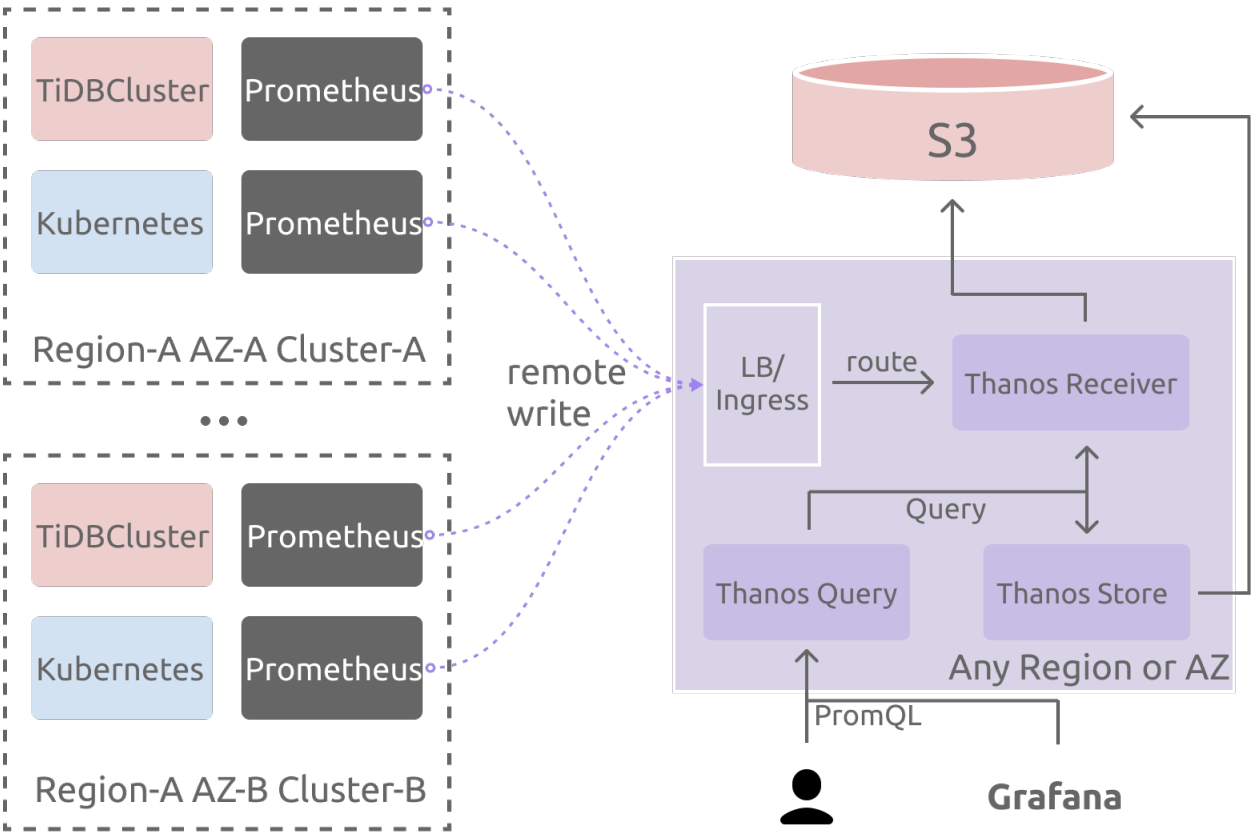


Figure 2: push-thanos-receive.png

5.4.1.3 Deploy TidbMonitor

1. According to the Kubernetes cluster that the TiDB cluster is deployed in, set the following environment variables:

    - `cluster_name`: the TiDB cluster name.
    - `cluster_namespace`: the TiDB cluster namespace.
    - `kubernetes_cluster_name`: the custom Kubernetes cluster name, which is used in the `externallabels` of Prometheus.
    - `storageclass_name`: the storage of the current Kubernetes cluster.
    - `remote_write_url`: the host of the `thanos-receiver` component, or the host of other components compatible with the Prometheus remote write API.

```
cluster_name="cluster1"
cluster_namespace="pingcap"
kubernetes_cluster_name="kind-cluster-1"
storageclass_name="local-storage"
remote_write_url="http://thanos-receiver:19291/api/v1/receive"
```

2. Create a `TidbMonitor` component by running the following command:

```
cat << EOF | kubectl apply -n ${cluster_namespace} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ${cluster_name}
spec:
  clusters:
  - name: ${cluster_name}
    namespace: ${cluster_namespace}
  externalLabels:
    # k8s_cluster indicates the k8s cluster name, you can change
    # the label's name on your own, but you should notice that the
    # "cluster" label has been used by the TiDB metrics already.
    # For more information, please refer to the issue
    # https://github.com/pingcap/tidb-operator/issues/4219.
    k8s_cluster: ${kubernetes_cluster_name}
    # add other meta labels here
    #region: us-east-1
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.4.0
  persistent: true
  storage: 100Gi
  storageClassName: ${storageclass_name}
  prometheus:
    baseImage: prom/prometheus
    logLevel: info
    remoteWrite:
    - url: ${remote_write_url}
    retentionTime: 2h
    version: v2.27.1
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
EOF
```

5.4.2 Pull data from Prometheus

The pull method pulls monitoring data from Prometheus instances in different Kubernetes clusters and aggregates the data into a global view. This section describes how to pull data using Thanos Query and using Prometheus Federation.

5.4.2.1 Using Thanos Query

In the example of this section, one Thanos Sidecar is deployed for each Prometheus (`TidbMonitor`) component. The Thanos query component is used for aggregation queries. If you do not need long-term storage, you can skip the deployment of some components such as thanos-store and S3.

5.4.2.1.1 Prerequisites

You need to configure the network and DNS of the Kubernetes clusters so that the Kubernetes clusters meet the following conditions:

- The Thanos Query component has access to the Pod IP of the Prometheus (`TidbMonitor`) component in each Kubernetes cluster.
- The Thanos Query component has access to the Pod FQDN of the Prometheus (`TidbMonitor`) component in each Kubernetes cluster.

For the deployment instructions of Thanos Query, refer to kube-thanos and the example.

5.4.2.1.2 Architecture

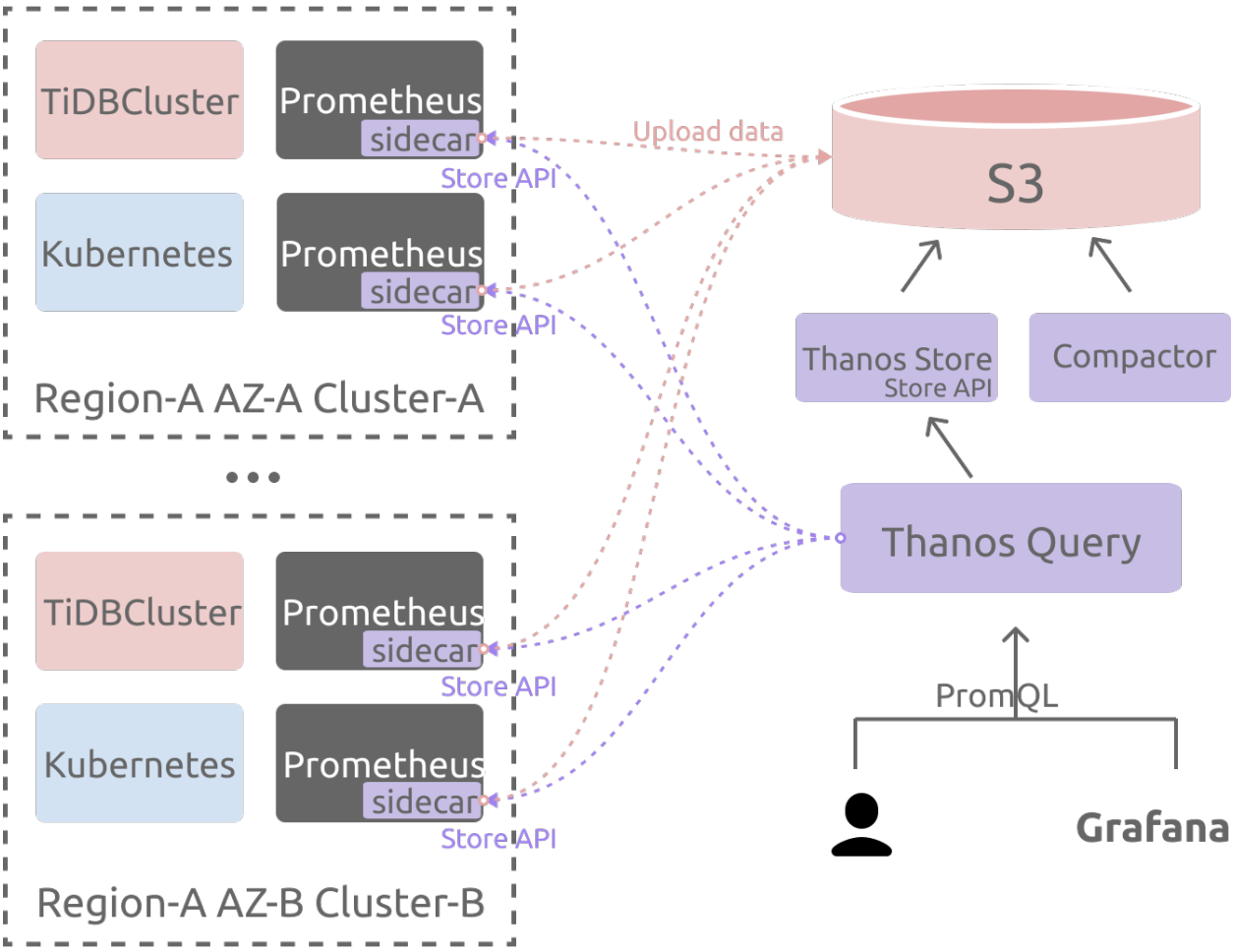


Figure 3: pull-thanos-query.png

5.4.2.1.3 Deploy TidbMonitor

1. According to the Kubernetes cluster that the TiDB cluster is deployed in, set the following environment variables:

    - `cluster_name`: the TiDB cluster name.
    - `cluster_namespace`: the TiDB cluster namespace.
    - `kubernetes_cluster_name`: the custom Kubernetes cluster name, which is used in the `externallabels` of Prometheus.
    - `cluster_domain`: the current cluster's cluster domain.
    - `storageclass_name`: the storage of the current Kubernetes cluster.

```
cluster_name="cluster1"
cluster_namespace="pingcap"
kubernetes_cluster_name="kind-cluster-1"
storageclass_name="local-storage"
```

```
cluster_domain="svc.local"
```

2. Create a `TidbMonitor` component by running the following command:

```
cat <<EOF | kubectl apply -n ${cluster_namespace} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ${cluster_name}
spec:
  clusters:
  - name: ${cluster_name}
    namespace: ${cluster_namespace}
  externalLabels:
    # k8s_cluster indicates the k8s cluster name, you can change
    # the label's name on your own, but you should notice that the
    # "cluster" label has been used by the TiDB metrics already.
    # For more information, please refer to the issue
    # https://github.com/pingcap/tidb-operator/issues/4219.
    k8s_cluster: ${kubernetes_cluster_name}
    # add other meta labels here
    #region: us-east-1
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.4.0
  persistent: true
  storage: 20Gi
  storageClassName: ${storageclass_name}
  prometheus:
    baseImage: prom/prometheus
    logLevel: info
    version: v2.27.1
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  thanos:
    baseImage: quay.io/thanos/thanos
    version: v0.22.0
    #enable config below if long-term storage is needed.
    #objectStorageConfig:
    #  key: objectstorage.yaml
    #  name: thanos-objectstorage
  imagePullPolicy: IfNotPresent
EOF
```

3. Configure Thanos Query Stores:

   You can specify the store nodes by the static service discovery method. In the Thanos Query command line's starting parameters, add `--store=${cluster_name}-prometheus.${cluster_namespace}.svc.${cluster_domain}:10901` to specify the store node. Replace the variables with the actual values.

   If you use other service discovery methods, refer to thanos-service-discovery.

5.4.2.2 Using Prometheus Federation

In the example of this section, the Federation Prometheus server is used as an entry point for unified storage and query. This method is only considered for small data volumes.

5.4.2.2.1 Prerequisites

You need to configure the network and DNS of the Kubernetes clusters so that the Kubernetes clusters meet the following conditions:

- The Federation Prometheus component has access to the Pod IP of the Prometheus (`TidbMonitor`) component in each Kubernetes cluster.
- The Federation Prometheus component has access to the Pod FQDN of the Prometheus (`TidbMonitor`) component in each Kubernetes cluster.

5.4.2.2.2 Architecture

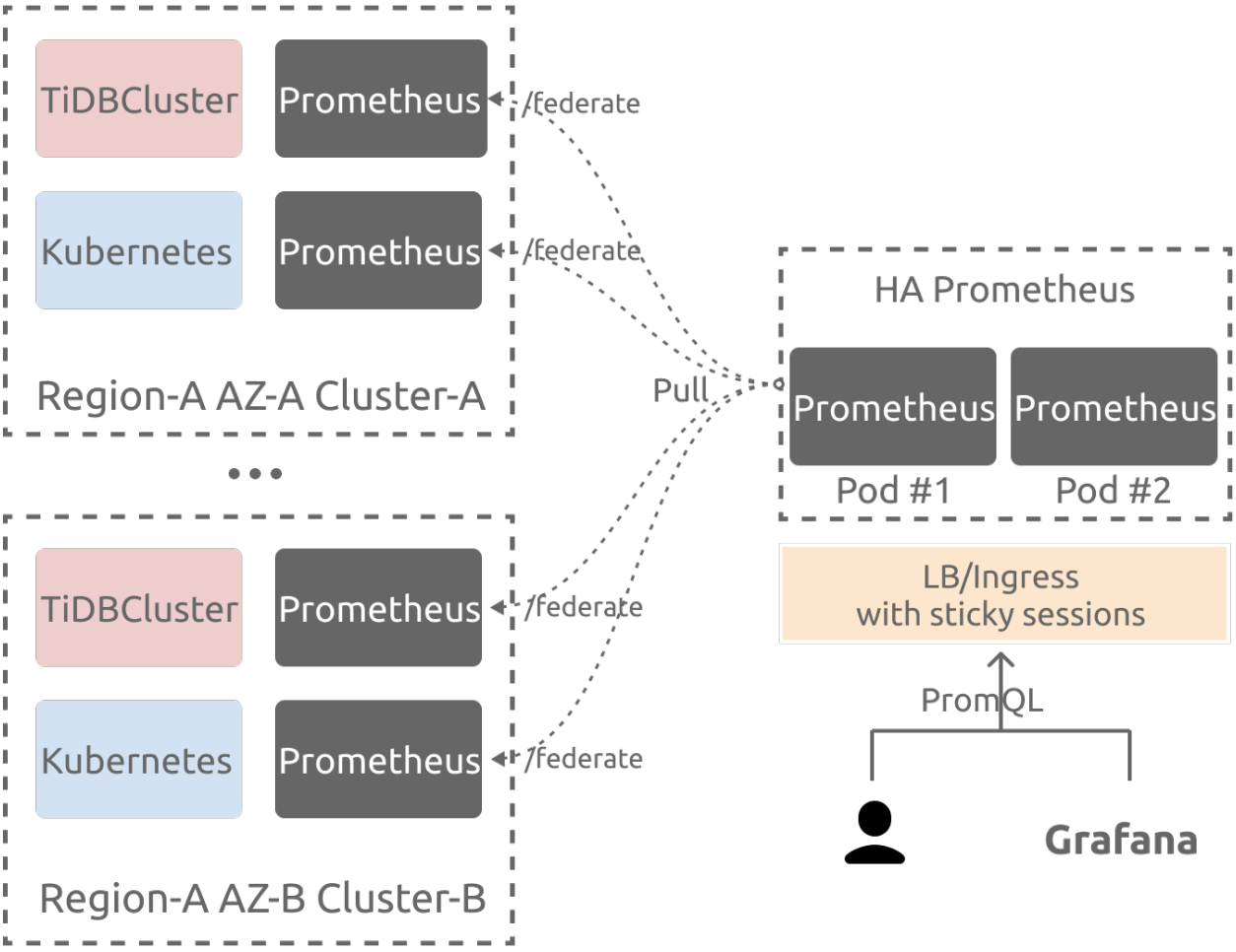


Figure 4: pull-prom-federation.png

5.4.2.2.3 Deploy TidbMonitor

1. According to the Kubernetes cluster that the TiDB cluster is deployed in, set the following environment variables:

    - `cluster_name`: the TiDB cluster name.
    - `cluster_namespace`: the TiDB cluster namespace.
    - `kubernetes_cluster_name`: the custom Kubernetes cluster name, which is used in the `externallabels` of Prometheus.
    - `storageclass_name`: the storage of the current Kubernetes cluster.

```
cluster_name="cluster1"
cluster_namespace="pingcap"
kubernetes_cluster_name="kind-cluster-1"
storageclass_name="local-storage"
```

2. Create a `TidbMonitor` component by running the following command:

```
cat << EOF | kubectl apply -n ${cluster_namespace} -f -
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: ${cluster_name}
spec:
  clusters:
  - name: ${cluster_name}
    namespace: ${cluster_namespace}
  externalLabels:
    # k8s_cluster indicates the k8s cluster name, you can change
    # the label's name on your own, but you should notice that the
    # "cluster" label has been used by the TiDB metrics already.
    # For more information, please refer to the issue
    # https://github.com/pingcap/tidb-operator/issues/4219.
    k8s_cluster: ${kubernetes_cluster_name}
    # add other meta labels here
    #region: us-east-1
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.4.0
  persistent: true
  storage: 20Gi
  storageClassName: ${storageclass_name}
  prometheus:
    baseImage: prom/prometheus
    logLevel: info
    version: v2.27.1
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  imagePullPolicy: IfNotPresent
EOF
```

5.4.2.2.4 Configure Federation Prometheus

For details on Federation, refer to Federation. After the deployment of Prometheus, you need to modify the Prometheus configuration, and add the host information of the Prometheus (`TidbMonitor`) to be aggregated.

```
scrape_configs:
  - job_name: 'federate'
    scrape_interval: 15s
```

```
honor_labels: true
metrics_path: '/federate'

params:
  'match[]':
    - '{__name__=~".+"}'

static_configs:
  - targets:
    - 'source-prometheus-1:9090'
    - 'source-prometheus-2:9090'
    - 'source-prometheus-3:9090'
```

5.4.3 Visualize monitoring data using Grafana

After collecting data using Prometheus, you can visualize multi-cluster monitoring data using Grafana.

1. Obtain the Grafana dashboard configuration files related to TiDB components by running the following command:

```
# set tidb version here
version=v8.5.0
docker run --rm -i -v ${PWD}/dashboards:/dashboards/ pingcap/tidb-
    ↪ monitor-initializer:${version} && \
cd dashboards
```

> **Note:**
>
> In the command above, `${version}` is the version of the Initializer image, which should be consistent with the TiDB version. Currently, only `v6 ↪ .0.0` and later versions of the Initializer image support **multiple Kubernetes clusters** monitoring.

   After running the command above, you can view all the dashboard JSON files in the current directory.

2. Refer to the Grafana documentation to configure the Prometheus data source.

   To keep the configuration consistent with the dashboard JSON files obtained in the previous step, you need to set the `Name` field of the data source to `tidb-cluster`. If you want to use the existing data source, run the following command to replace the data source name in the dashboard JSON files. In the command, `$DS_NAME` is the name of the data source.

```
# define your datasource name here.
DS_NAME=thanos
sed -i 's/"datasource": "tidb-cluster"/"datasource": "$DS_NAME"/g' *.
    ↪ json
```

3. Refer to the Grafana documentation to import dashboards into Grafana.

5.5 Enable Dynamic Configuration for TidbMonitor

This document describes how to enable dynamic configuration for TidbMonitor.

TidbMonitor supports monitoring across multiple clusters and shards. Without dynamic configuration, when the Prometheus configurations, rules, or targets are changed, such changes only take effect after a restart. If you are monitoring a large dataset, after the restart, it might take a long time to recover the Prometheus snapshot data.

With dynamic configuration enabled, any configuration change of TidbMonitor takes effect immediately.

5.5.1 Enable the dynamic configuration feature

To enable the dynamic configuration feature, configure `prometheusReloader` in the `spec` field of TidbMonitor. For example:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: monitor
spec:
  clusterScoped: true
  clusters:
    - name: ns1
      namespace: ns1
    - name: ns2
      namespace: ns2
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
  imagePullPolicy: IfNotPresent
```

After you modify the `prometheusReloader` configuration, TidbMonitor restarts automatically. After the restart, the dynamic configuration feature is enabled. All configuration changes related to Prometheus are dynamically updated.

For more examples, refer to monitor-dynamic-configmap.

5.5.2 Disable the dynamic configuration feature

To disable the dynamic configuration feature, remove the `prometheusReloader` configuration from the `spec` field of TidbMonitor.

5.6 Enable Shards for TidbMonitor

This document describes how to use shards for TidbMonitor.

5.6.1 Shards

TidbMonitor collects monitoring data for a single TiDB cluster or multiple TiDB clusters. When the amount of monitoring data is large, the computing capacity of one TidbMonitor might hit a bottleneck. In this case, it is recommended to use shards of Prometheus Modulus. This feature performs `hashmod` on `__address__` to divide the monitoring data of multiple targets (`Targets`) into multiple TidbMonitor Pods.

To use shards for TidbMonitor, you need a data aggregation plan. The Thanos method is recommended.

5.6.2 Enable shards

To enable shards for TidbMonitor, you need to specify the `shards` field. For example:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbMonitor
metadata:
  name: monitor
spec:
  replicas: 1
  shards: 2
  clusters:
    - name: basic
  prometheus:
    baseImage: prom/prometheus
    version: v2.27.1
  initializer:
    baseImage: pingcap/tidb-monitor-initializer
    version: v5.2.1
  reloader:
    baseImage: pingcap/tidb-monitor-reloader
    version: v1.0.1
  prometheusReloader:
    baseImage: quay.io/prometheus-operator/prometheus-config-reloader
    version: v0.49.0
```

```
  imagePullPolicy: IfNotPresent
```

Note:

- The number of Pods corresponding to TidbMonitor is the product of `replicas` and `shards`. For example, when `replicas` is `1` and `shards` is 2, TiDB Operator creates 2 TidbMonitor Pods.
- After `shards` is changed, `Targets` are reallocated. However, the monitoring data already stored on the Pods is not reallocated.

For details on the configuration, refer to shards example.

6 Migrate

6.1 Import Data

This document describes how to import data into a TiDB cluster on Kubernetes using TiDB Lightning.

In Kubernetes, the tidb-lightning is in a separate Helm chart and deployed as a `Job`.

TiDB Lightning supports two backends: `Local-backend`, and `TiDB-backend`. For the differences of these backends and how to choose backends, see TiDB Lightning Backends.

- 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

6.1.1 Deploy TiDB Lightning

6.1.1.1 Step 1. Configure TiDB Lightning

Use the following command to save the default configuration of TiDB Lightning to the `tidb-lightning-values.yaml` file:

```
helm inspect values pingcap/tidb-lightning --version=${chart_version} > tidb
    ↪ -lightning-values.yaml
```

Configure the `backend` field in the configuration file depending on your needs. The optional values are `local` and `tidb`.

```
## The delivery backend used to import data (valid options include `local`
   ↪ and `tidb`).
## If set to `local`, then the following `sortedKV` should be set.
backend: local
```

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.

```
## For `local` backend, an extra PV is needed for local KV sorting.
sortedKV:
  storageClassName: local-storage
  storage: 100Gi
```

6.1.1.1.1 Configure checkpoint

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.

6.1.1.1.2 Configure TLS

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 generate 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"
```

6.1.1.2 Step 2. Configure the data source

The tidb-lightning Helm chart supports both local and remote data sources. The three types of data sources correspond to three modes: local, remote, and ad hoc. The three modes cannot be used together. You can only configure one mode.

6.1.1.2.1 Local

In the local mode, tidb-lightning reads the backup data from a directory in one of the Kubernetes node.

```
dataSource:
  local:
    nodeName: kind-worker3
    hostPath: /data/export-20190820
```

The descriptions of the related fields are as follows:

- `dataSource.local.nodeName`: the node name that the directory is located at.
- `dataSource.local.hostPath`: the path of the backup data. The path must contain a file named `metadata`.

6.1.1.2.2 Remote

Unlike the local mode, the remote mode uses rclone to download the backup tarball file or the backup directory 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. 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}
    ```
````

2. Configure the `dataSource` field. For example:

```
dataSource:
  remote:
    rcloneImage: rclone/rclone:1.55.1
    storageClassName: local-storage
    storage: 100Gi
    secretName: cloud-storage-secret
    path: s3:bench-data-us/sysbench/sbtest_16_1e7.tar.gz
    # directory: s3:bench-data-us
```

The descriptions of the related fields are as follows:

- `dataSource.remote.storageClassName`: the name of StorageClass used to create PV.
- `dataSource.remote.secretName`: the name of the Secret created in the previous step.
- `dataSource.remote.path`: If the backup data is packaged as a tarball file, use this field to indicate the path to the tarball file.
- `dataSource.remote.directory`: If the backup data is in a directory, use this field to specify the path to the directory.

6.1.1.2.3 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.

For example:

```
dataSource:
  adhoc:
    pvcName: tidb-cluster-scheduled-backup
    backupName: scheduled-backup-20190822-041004
```

The descriptions of the related fields are as follows:

- `dataSource.adhoc.pvcName`: the PVC name used in restoring data from remote storage. The PVC must be deployed in the same namespace as Tidb-Lightning.
- `dataSource.adhoc.backupName`: the name of the original backup data, such as: `backup-2020-12-17T10:12:51Z` (Does not contain the '. tgz' suffix of the compressed file name on network storage).

6.1.1.3 Step 3. 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`.

6.1.2 Destroy 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 tidb-lightning, execute the following command:

```
helm uninstall ${release_name} -n ${namespace}
```

6.1.3 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.

6.2 Migrate from MySQL

6.2.1 Deploy DM on Kubernetes

TiDB Data Migration (DM) is an integrated data migration task management platform that supports the full data migration and the incremental data replication from MySQL/-MariaDB into TiDB. This document describes how to deploy DM on Kubernetes using TiDB Operator and how to migrate MySQL data to TiDB cluster using DM.

6.2.1.1 Prerequisites

- Complete deploying TiDB Operator.

> **Note:**
>
> Make sure that the TiDB Operator version >= 1.2.0.

6.2.1.2 Configure DM deployment

To configure the DM deployment, you need to configure the `DMCluster` Custom Resource (CR). For the complete configurations of the `DMCluster` CR, refer to the DMCluster example and API documentation. Note that you need to choose the example and API of the current TiDB Operator version.

6.2.1.2.1 Cluster name

Configure the cluster name by changing the `metadata.name` in the `DMCluster` CR.

6.2.1.2.2 Version

Usually, components in a cluster are in the same version. It is recommended to configure only `spec.<master/worker>.baseImage` and `spec.version`. If you need to deploy different versions for different components, configure `spec.<master/worker>.version`.

The formats of the related parameters are as follows:

- `spec.version`: the format is `imageTag`, such as `v8.5.0`.
- `spec.<master/worker>.baseImage`: the format is `imageName`, such as `pingcap/dm`.
- `spec.<master/worker>.version`: the format is `imageTag`, such as `v8.5.0`.

TiDB Operator only supports deploying DM 2.0 and later versions.

6.2.1.2.3 Cluster

Configure DM-master

DM-master is an indispensable component of the DM cluster. You need to deploy at least three DM-master Pods if you want to achieve high availability.

You can configure DM-master parameters by `spec.master.config` in `DMCluster` CR. For complete DM-master configuration parameters, refer to DM-master Configuration File.

```
apiVersion: pingcap.com/v1alpha1
kind: DMCluster
metadata:
  name: ${dm_cluster_name}
  namespace: ${namespace}
spec:
  version: v8.5.0
  configUpdateStrategy: RollingUpdate
  pvReclaimPolicy: Retain
  discovery: {}
  master:
    baseImage: pingcap/dm
    maxFailoverCount: 0
    imagePullPolicy: IfNotPresent
    service:
      type: NodePort
      # Configures masterNodePort when you need to expose the DM-master
          ↪ service to a fixed NodePort
      # masterNodePort: 30020
    replicas: 1
    storageSize: "10Gi"
    requests:
      cpu: 1
    config: |
```

```
    rpc-timeout = "40s"
```

Configure DM-worker

You can configure DM-worker parameters by `spec.worker.config` in `DMCluster` CR. For complete DM-worker configuration parameters, refer to DM-worker Configuration File.

```
apiVersion: pingcap.com/v1alpha1
kind: DMCluster
metadata:
  name: ${dm_cluster_name}
  namespace: ${namespace}
spec:
  ...
  worker:
    baseImage: pingcap/dm
    maxFailoverCount: 0
    replicas: 1
    storageSize: "100Gi"
    requests:
      cpu: 1
    config: |
      keepalive-ttl = 15
```

6.2.1.2.4 Topology Spread Constraint

By configuring `topologySpreadConstraints`, you can make pods evenly spread in different topologies. For instructions about configuring `topologySpreadConstraints`, see Pod Topology Spread Constraints.

You can either configure `topologySpreadConstraints` at a cluster level (`spec.topologySpreadConstraints`) for all components or at a component level (such as `spec.tidb.topologySpreadConstraints`) for specific components.

The following is an example configuration:

```
topologySpreadConstrains:
- topologyKey: kubernetes.io/hostname
- topologyKey: topology.kubernetes.io/zone
```

The example configuration can make pods of the same component evenly spread on different zones and nodes.

Currently, `topologySpreadConstraints` only supports the configuration of the `topologyKey` field. In the pod spec, the above example configuration will be automatically expanded as follows:

```
topologySpreadConstrains:
```

```
- topologyKey: kubernetes.io/hostname
  maxSkew: 1
  whenUnsatisfiable: DoNotSchedule
  labelSelector: <object>
- topologyKey: topology.kubernetes.io/zone
  maxSkew: 1
  whenUnsatisfiable: DoNotSchedule
  labelSelector: <object>
```

6.2.1.3 Deploy the DM cluster

After configuring the yaml file of the DM cluster in the above steps, execute the following command to deploy the DM cluster:

```
kubectl apply -f ${dm_cluster_name}.yaml -n ${namespace}
```

If the server does not have an external network, you need to download the Docker image used by the DM cluster and upload the image to the server, and then execute `docker load` to install the Docker image on the server:

1. Deploy a DM cluster requires the following Docker image (assuming the version of the DM cluster is v8.5.0):

   ```
   pingcap/dm:v8.5.0
   ```

2. To download the image, execute the following command:

   ```
   docker pull pingcap/dm:v8.5.0
   docker save -o dm-v8.5.0.tar pingcap/dm:v8.5.0
   ```

3. Upload the Docker image to the server, and execute `docker load` to install the image on the server:

   ```
   docker load -i dm-v8.5.0.tar
   ```

After deploying the DM cluster, execute the following command to view the Pod status:

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

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

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

6.2.1.4 Access the DM cluster on Kubernetes

To access DM-master in the pod within a Kubernetes cluster, use the DM-master service domain name `${cluster_name}-dm-master.${namespace}`.

To access the DM cluster outside a Kubernetes cluster, expose the DM-master port by editing the `spec.master.service` field configuration in the `DMCluster` CR.

```
spec:
  ...
  master:
    service:
      type: NodePort
```

You can access the DM-master service via the address of `${kubernetes_node_ip}:${node_port}`.

For more service exposure methods, refer to Access the TiDB Cluster.

6.2.1.5 What's next

- To migrate MySQL data to your TiDB cluster using DM on Kubernetes, see Migrate MySQL Data to TiDB Cluster Using DM.
- To enable TLS between components of the DM cluster on Kubernetes, see Enable TLS for DM.

6.2.2 Use DM on Kubernetes

TiDB Data Migration (DM) is an integrated data migration task management platform that supports the full data migration and the incremental data replication from MySQL/-MariaDB into TiDB. This document describes how to migrate MySQL data to TiDB cluster using DM on Kubernetes.

6.2.2.1 Prerequisites

- Complete deploying TiDB Operator.
- Complete deploying DM on Kubernetes.

> **Note:**
>
> Make sure that the TiDB Operator version >= 1.2.0.

6.2.2.2 Enable DM data migration tasks

You can access the DM-master service using dmctl in the following two methods:

Method #1: Attach to the DM-master or DM-worker Pod to use the built-in `dmctl` in the image.

Method #2: Expose the DM-master service by accessing the DM cluster on Kubernetes and use `dmctl` outside the pods to access the exposed DM-master service.

It is recommended to use **Method #1** for migration. The following steps take **Method #1** as an example to introduce how to start a DM data migration task.

The differences between Method #1 and Method #2 are that the file locations of `source.yaml` and `task.yaml` are different, and that in Method #2 you need to configure the exposed DM-master service address in the `master-addr` configuration item of `dmctl`.

6.2.2.2.1 Get into the Pod

Attach to the DM-master Pod by executing the following command:

```
kubectl exec -ti ${dm_cluster_name}-dm-master-0 -n ${namespace} -- /bin/sh
```

6.2.2.2.2 Create data source

1. Write MySQL-1 related information to `source1.yaml` file, which can refer to Create data source.
2. Configure the `from.host` in the `source1.yaml` file as the MySQL host address that the Kubernetes cluster can access internally.
3. Configure the `relay-dir` in the `source1.yaml` file as a subdirectory of the persistent volume in the Pod mount `/var/lib/dm-worker` directory. For example, `/var/lib/dm-worker/relay`.
4. After you prepare the `source1.yaml` file, load the MySQL-1 data source into the DM cluster by executing the following command:

   ```
   /dmctl --master-addr ${dm_cluster_name}-dm-master:8261 operate-source create source1.yaml
   ```

5. For MySQL-2 and other data sources, use the same method to modify the relevant information in the data source `yaml` file and execute the same dmctl command to load the corresponding data source into the DM cluster.

6.2.2.2.3 Configure migration tasks

1. Edit task configuration file `task.yaml`, which can refer to Configure the data migration task.
2. Configure the `target-database.host` in `task.yaml` as the TiDB host address that the Kubernetes cluster can access internally. If the cluster is deployed by TiDB Operator, configure the host as `${tidb_cluster_name}-tidb.${namespace}`.
3. In the `task.yaml` file, take the following steps:
    - Add the `loaders.${customized_name}.dir` field as the import and export directory for the full volume data, where `${customized_name}` is a name that you can customize.
    - Configure the `loaders.${customized_name}.dir` field as the subdirectory of the persistent volume in the Pod `/var/lib/dm-worker` directory. For example, `/var/lib/dm-worker/dumped_data`.
    - Reference `${customized_name}` in the instance configuration. For example, `mysql-instances[0].loader-config-name: "{customized_name}"`.

6.2.2.2.4 Start/Check/Stop the migration tasks

Refer to the corresponding steps in Migrate Data Using DM and fill in the master-addr as `${dm_cluster_name}-dm-master:8261`.

6.3 Migrate TiDB to Kubernetes

This document describes how to migrate a TiDB cluster deployed in the physical or virtual machine to a Kubernetes cluster, without using any backup and restore tool.

6.3.1 Prerequisites

- The physical or virtual machines outside Kubernetes have network access to the Pods in Kubernetes.
- The physical or virtual machines outside Kubernetes can resolve the domain name of the Pods in Kubernetes. (See Step 1 for configuration details.)
- The cluster to be migrated (that is, the source cluster) does not enable TLS between components.

6.3.2 Step 1: Configure DNS service in all nodes of the cluster to be migrated

1. Get the Pod IP address list of the endpoints of the CoreDNS or kube-dns service of the Kubernetes cluster:

```
kubectl describe svc/kube-dns -n kube-system
```

2. Modify the `/etc/resolv.conf` configuration of the source cluster node, and add the following content to the configuration file:

```
search default.svc.cluster.local svc.cluster.local cluster.local
nameserver <CoreDNS Pod_IP_1>
nameserver <CoreDNS Pod_IP_2>
nameserver <CoreDNS Pod_IP_n>
```

3. Test whether the node can successfully resolve the domain name of the Pods in Kubernetes:

```
$ ping basic-pd-2.basic-pd-peer.blade.svc
PING basic-pd-2.basic-pd-peer.blade.svc (10.24.66.178) 56(84) bytes of
    ↪ data.
64 bytes from basic-pd-2.basic-pd-peer.blade.svc (10.24.66.178):
    ↪ icmp_seq=1 ttl=61 time=0.213 ms
64 bytes from basic-pd-2.basic-pd-peer.blade.svc (10.24.66.178):
    ↪ icmp_seq=2 ttl=61 time=0.175 ms
64 bytes from basic-pd-2.basic-pd-peer.blade.svc (10.24.66.178):
    ↪ icmp_seq=3 ttl=61 time=0.188 ms
64 bytes from basic-pd-2.basic-pd-peer.blade.svc (10.24.66.178):
    ↪ icmp_seq=4 ttl=61 time=0.157 ms
```

6.3.3 Step 2: Create a TiDB cluster on Kubernetes

1. Get the PD node address and port of the source cluster via PD Control:

```
pd-ctl -u http://<address>:<port> member | jq '.members | .[] | .
    ↪ client_urls'
```

2. Create the target TiDB cluster on Kubernetes, which must have at least 3 TiKV nodes. Specify the PD node address of the source cluster in the `spec.pdAddresses` field (starting with `http://`):

```
spec
  ...
  pdAddresses:
  - http://pd1_addr:port
  - http://pd2_addr:port
  - http://pd3_addr:port
```

3. Confirm that the source cluster and the target cluster compose of a new cluster that runs normally:

    - Get the number and state of stores in the new cluster:

```
# Get the number of stores
pd-ctl -u http://<address>:<port> store | jq '.count'
# Get the state of stores
pd-ctl -u http://<address>:<port> store | jq '.stores | .[] | .
    ↪ store.state_name'
```

- Access the TiDB cluster on Kubernetes via MySQL client.

6.3.4 Step 3: Scale in the TiDB nodes of the source cluster

Remove all TiDB nodes of the source cluster:

- If the source cluster is deployed using TiUP, refer to Scale in a TiDB/PD/TiKV cluster.
- If the source cluster is deployed using TiDB Ansible, refer to Decrease the capacity of a TiDB node.

Note:

If you access the source TiDB cluster via load balancer or database middleware, you need to first modify the configuration to route your application traffic to the target TiDB cluster. Otherwise, your application might be affected.

6.3.5 Step 4: Scale in the TiKV nodes of the source cluster

Remove all TiKV nodes of the source cluster:

- If the source cluster is deployed using TiUP, refer to Scale in a TiDB/PD/TiKV cluster.
- If the source cluster is deployed using TiDB Ansible, refer to Decrease the capacity of a TiKV node.

Note:

- You need to scale in the TiKV nodes one by one. Wait until the store state of one TiKV node becomes "tombstone" and then scale in the next TiKV node.
- You can view the store state using PD Control.

6.3.6 Step 5: Scale in the PD nodes of the source cluster

Remove all PD nodes of the source cluster:

- If the source cluster is deployed using TiUP, refer to Scale in a TiDB/PD/TiKV cluster.
- If the source cluster is deployed using TiDB Ansible, refer to Decrease the capacity of a PD node.

6.3.7 Step 6: Delete the `spec.pdAddresses` field

To avoid confusion for further operations on the cluster, it is recommended that you delete the `spec.pdAddresses` field in the new cluster after the migration.

7 Manage

7.1 Secure

7.1.1 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`.

- To skip TLS authentication for internal components that serve as the MySQL client (such as TidbInitializer, Dashboard, Backup, and Restore), you can add the `tidb.tidb.pingcap.com/skip-tls-when-connect-tidb="true"` annotation to the cluster's corresponding `TidbCluster`.
- To disable the client CA certificate authentication on the TiDB server, you can set `.spec.tidb.tlsClient.disableClientAuthn` to `true`. This means skipping setting the `ssl-ca` parameter when you configure TiDB server to enable secure connections.
- To skip the CA certificate authentication for internal components that serve as the MySQL client, you can set `.spec.tidb.tlsClient.skipInternalClientCA` to `true`.

> **Note:**
> For an existing cluster, if you change `.spec.tidb.tlsClient.enabled` from `false` to `true`, the TiDB Pods will be rolling restarted.

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.

7.1.1.1 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`.

7.1.1.1.1 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.

7.1.1.1.2 Using `cert-manager`

1. Install `cert-manager`.

   Refer to cert-manager installation on 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/v1
kind: Issuer
metadata:
  name: ${cluster_name}-selfsigned-ca-issuer
  namespace: ${namespace}
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${cluster_name}-ca
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-ca-secret
  commonName: "TiDB CA"
  isCA: true
  duration: 87600h # 10yrs
  renewBefore: 720h # 30d
  issuerRef:
    name: ${cluster_name}-selfsigned-ca-issuer
    kind: Issuer
---
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: ${cluster_name}-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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-server-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-server-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB Server"
  usages:
    - server auth
  dnsNames:
    - "${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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - 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 (when using Dumpling)
   - Restore (when using TiDB Lightning)

   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/v1
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
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB Initializer client"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-tidb-issuer
```

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

```
metadata:
  name: ${cluster_name}-restore-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-restore-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - 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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-lightning-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-lightning-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - 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.

7.1.1.2 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: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Retain
  pd:
    baseImage: pingcap/pd
```

```
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "10Gi"
    config: {}
    tlsClientSecretName: ${cluster_name}-pd-dashboard-client-secret
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 1
    requests:
      storage: "100Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    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
  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
  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
```

7.1.1.3 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 --comments -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.

7.1.2 Enable TLS between TiDB Components

This document describes how to enable Transport Layer Security (TLS) between components of the TiDB cluster on 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/TiProxy/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`.

> **Note:**
>
> After the cluster is created, do not modify this field; otherwise, the cluster will fail to upgrade. If you need to modify this field, delete the cluster and create a new one.

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.

7.1.2.1 Generate certificates for components of the TiDB cluster

This section describes how to issue certificates using two methods: `cfssl` and `cert-manager`.

7.1.2.1.1 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"
  ],
...
```

Note:

Starting from v8.0.0, PD supports the microservice mode (experimental). To deploy PD microservices in your cluster, it is unnecessary to generate certificates for each component of PD microservices. Instead, you only need to add the host configurations for microservices to the `hosts` field of the `pd-server.json` file. Taking the `scheduling` microservice as an example, you need to configure the following items:

```
...
  "CN": "TiDB",
  "hosts": [
    "127.0.0.1",
    "::1",
    "${cluster_name}-pd",
    ...
    "*.${cluster_name}-pd-peer.${namespace}.svc",
    // The following are host configurations for the `
       ↪ scheduling` microservice
    "${cluster_name}-scheduling",
    "${cluster_name}-scheduling.${cluster_name}",
    "${cluster_name}-scheduling.${cluster_name}.svc",
    "${cluster_name}-scheduling-peer",
    "${cluster_name}-scheduling-peer.${cluster_name}",
    "${cluster_name}-scheduling-peer.${cluster_name}.
       ↪ svc",
    "*.${cluster_name}-scheduling-peer",
    "*.${cluster_name}-scheduling-peer.${cluster_name
       ↪ }",
    "*.${cluster_name}-scheduling-peer.${cluster_name}.
       ↪ 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
```

- TiProxy

1. Generate the default `tiproxy-server.json` file:

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

2. Edit this file to change the `CN` and `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${cluster_name}-tiproxy",
      "${cluster_name}-tiproxy.${namespace}",
      "${cluster_name}-tiproxy.${namespace}.svc",
      "${cluster_name}-tiproxy-peer",
      "${cluster_name}-tiproxy-peer.${namespace}",
      "${cluster_name}-tiproxy-peer.${namespace}.svc",
      "*.${cluster_name}-tiproxy-peer",
      "*.${cluster_name}-tiproxy-peer.${namespace}",
      "*.${cluster_name}-tiproxy-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 TiProxy server-side certificate:

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.
    ↪ json -profile=internal tiproxy-server.json | cfssljson -
    ↪ bare tiproxy-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 TiProxy cluster certificate Secret:

```
kubectl create secret generic ${cluster_name}-tiproxy-cluster-
    ↪ secret --namespace=${namespace} --from-file=tls.crt=tiproxy-
    ↪ server.pem --from-file=tls.key=tiproxy-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.

7.1.2.1.2 Using `cert-manager`

1. Install `cert-manager`.

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

```
  name: ${cluster_name}-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/v1
   kind: Certificate
   metadata:
     name: ${cluster_name}-pd-cluster-secret
     namespace: ${namespace}
   spec:
     secretName: ${cluster_name}-pd-cluster-secret
     duration: 8760h # 365d
     renewBefore: 360h # 15d
     subject:
       organizations:
       - PingCAP
     commonName: "TiDB"
     usages:
       - server auth
       - client auth
     dnsNames:
     - "${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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-tikv-cluster-secret
```

```
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tikv-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-tidb-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tidb-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-pump-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-pump-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${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/v1
```

```
kind: Certificate
metadata:
  name: ${cluster_name}-drainer-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-drainer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "*.${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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-drainer-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-drainer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - 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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-ticdc-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-ticdc-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - 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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-tiflash-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-tiflash-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-importer-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-importer-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-lightning-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-lightning-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - 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/v1
kind: Certificate
metadata:
  name: ${cluster_name}-cluster-client-secret
  namespace: ${namespace}
spec:
  secretName: ${cluster_name}-cluster-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
```

```
  - 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.

7.1.2.2 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 with a monitoring system and the Pump component:

   Create the `tidb-cluster.yaml` file:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
 name: ${cluster_name}
 namespace: ${namespace}
spec:
 tlsCluster:
   enabled: true
 version: v8.5.0
 timezone: UTC
 pvReclaimPolicy: Retain
 pd:
   baseImage: pingcap/pd
   maxFailoverCount: 0
   replicas: 1
   requests:
     storage: "10Gi"
   config:
     security:
       cert-allowed-cn:
         - TiDB
 tikv:
   baseImage: pingcap/tikv
   maxFailoverCount: 0
   replicas: 1
   requests:
     storage: "100Gi"
   config:
     security:
       cert-allowed-cn:
         - TiDB
 tidb:
   baseImage: pingcap/tidb
   maxFailoverCount: 0
   replicas: 1
   service:
     type: ClusterIP
   config:
     security:
       cluster-verify-cn:
         - TiDB
 pump:
   baseImage: pingcap/tidb-binlog
   replicas: 1
```

```
  requests:
    storage: "100Gi"
  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: v2.27.1
 grafana:
   baseImage: grafana/grafana
   version: 7.5.11
 initializer:
   baseImage: pingcap/tidb-monitor-initializer
   version: v8.5.0
 reloader:
   baseImage: pingcap/tidb-monitor-reloader
   version: v1.0.1
 prometheusReloader:
   baseImage: quay.io/prometheus-operator/prometheus-config-reloader
   version: v0.49.0
 imagePullPolicy: IfNotPresent
```

Execute `kubectl apply -f tidb-cluster.yaml` to create a TiDB cluster.

Note:

Starting from v8.0.0, PD supports the microservice mode (experimental). To deploy PD microservices, you need to configure `cert-allowed-cn` for each microservice. Taking the Scheduling service as an example, you need to make the following configurations:

- Update `pd.mode` to `ms`.
- Configure the `security` field for the `scheduling` microservice.

```
pd:
 baseImage: pingcap/pd
```

```
  maxFailoverCount: 0
  replicas: 1
  requests:
   storage: "10Gi"
  config:
   security:
     cert-allowed-cn:
       - TiDB
  mode: "ms"
 pdms:
 - name: "scheduling"
   baseImage: pingcap/pd
   replicas: 1
   config:
     security:
       cert-allowed-cn:
         - TiDB
```

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
  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 s3: provider: aws region: ${
↪ my_region} secretName: ${s3_secret} bucket: ${my_bucket} prefix:
↪  ${my_folder}
```

   Deploy Restore:

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

7.1.2.3 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 patch tc ${cluster_name} -n ${namespace} --type merge -p '{"
   ↪ spec":{"pd":{"mountClusterClientSecret": true},"tikv":{"
   ↪ mountClusterClientSecret": true}}}'
```

> **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
```

7.1.3 Enable TLS for DM

This document describes how to enable TLS between components of the DM cluster on Kubernetes and how to use DM to migrate data between MySQL/TiDB databases that enable TLS for the MySQL client.

7.1.3.1 Enable TLS between DM components

Starting from v1.2, TiDB Operator supports enabling TLS between components of the DM cluster on Kubernetes.

To enable TLS between components of the DM cluster, perform the following steps:

1. Generate certificates for each component of the DM cluster to be created:

    - A set of server-side certificates for the DM-master/DM-worker 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}-dm-client-secret`.

> **Note:**
> The Secret objects you created must follow the above naming convention. Otherwise, the deployment of the DM cluster will fail.

2. Deploy the cluster, and set `.spec.tlsCluster.enabled` to `true`.

> **Note:**
> After the cluster is created, do not modify this field; otherwise, the cluster will fail to upgrade. If you need to modify this field, delete the cluster and create a new one.

3. Configure `dmctl` to connect to the cluster.

Certificates can be issued in multiple methods. This document describes two methods. You can choose either of them to issue certificates for the DM 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.

7.1.3.1.1 Generate certificates for components of the DM cluster

This section describes how to issue certificates using two methods: `cfssl` and `cert-manager`.

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 DM cluster.

   - DM-master

     First, generate the default `dm-master-server.json` file:

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

     Then, edit this file to change the `CN` and `hosts` attributes:

     ```
     ...
         "CN": "TiDB",
         "hosts": [
           "127.0.0.1",
     ```

```
      "::1",
      "${cluster_name}-dm-master",
      "${cluster_name}-dm-master.${namespace}",
      "${cluster_name}-dm-master.${namespace}.svc",
      "${cluster_name}-dm-master-peer",
      "${cluster_name}-dm-master-peer.${namespace}",
      "${cluster_name}-dm-master-peer.${namespace}.svc",
      "*.${cluster_name}-dm-master-peer",
      "*.${cluster_name}-dm-master-peer.${namespace}",
      "*.${cluster_name}-dm-master-peer.${namespace}.svc"
    ],
...
```

`${cluster_name}` is the name of the DM cluster. `${namespace}` is the namespace in which the DM cluster is deployed. You can also add your customized `hosts`.

Finally, generate the DM-master server-side certificate:

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

- DM-worker

  First, generate the default `dm-worker-server.json` file:

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

  Then, edit this file to change the `CN` and `hosts` attributes:

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

  `${cluster_name}` is the name of the cluster. `${namespace}` is the namespace in which the DM cluster is deployed. You can also add your customized `hosts`.

Finally, generate the DM-worker server-side certificate:

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

6. Generate the client-side certificates:

   First, generate 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 DM cluster by executing the following command:

   - The DM-master cluster certificate Secret:

```
kubectl create secret generic ${cluster_name}-dm-master-cluster-
    ↪ secret --namespace=${namespace} --from-file=tls.crt=dm-
    ↪ master-server.pem --from-file=tls.key=dm-master-server-key.
    ↪ pem --from-file=ca.crt=ca.pem
```

   - The DM-worker cluster certificate Secret:

```
kubectl create secret generic ${cluster_name}-dm-worker-cluster-
    ↪ secret --namespace=${namespace} --from-file=tls.crt=dm-
    ↪ worker-server.pem --from-file=tls.key=dm-worker-server-key.
    ↪ pem --from-file=ca.crt=ca.pem
```

   - Client certificate Secret:

```
kubectl create secret generic ${cluster_name}-dm-client-secret --
    ↪ namespace=${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 DM-master/DM-worker server-side certificate to load when the server is started;
- One Secret object for their clients to connect.

Using `cert-manager`

1. Install `cert-manager`.

   Refer to cert-manager installation on Kubernetes for details.

2. Create an Issuer to issue certificates to the DM 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 `dm-cluster-issuer.yaml` file with the following content:

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

```
kind: Issuer
metadata:
  name: ${cluster_name}-dm-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 components of the DM cluster.

Finally, execute the following command to create an Issuer:

```
kubectl apply -f dm-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.

   Each component needs a server-side certificate, and all components need a shared client-side certificate for their clients.

   - The DM-master server-side certificate

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

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

`${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${cluster_name}-dm-master-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  * "${cluster_name}-dm-master"
  * "$cluster_{name} - dm - master.{namespace}"
  * "$cluster_{name} - dm - master.{namespace}.svc"
  * "${cluster_name}-dm-master-peer"
  * "$cluster_{name} - dm - master - peer.{namespace}"
  * "$cluster_{name} - dm - master - peer.{namespace}.svc"
  * "*.${cluster_name}-dm-master-peer"
  * "*.$cluster_{name} - dm - master - peer.{namespace}"
  * "*.$cluster_{name} - dm - master - 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}-dm-` ↪ `master-cluster-secret` Secret object to be used by the DM-master component of the DM cluster.

- The DM-worker server-side certificate

```
apiVersion: cert-manager.io/v1
kind: Certificate
```

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

`${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${cluster_name}-dm-worker-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  * "${cluster_name}-dm-worker"
  * "$cluster_name - dm - worker.${namespace}"
  * "$cluster_name - dm - worker.${namespace}.svc"
  * "${cluster_name}-dm-worker-peer"
  * "$cluster_name - dm - worker - peer.${namespace}"
  * "$cluster_name - dm - worker - peer.${namespace}.svc"
  * "*.${cluster_name}-dm-worker-peer"
  * "*.$cluster_name - dm - worker - peer.${namespace}"

    * "*.$cluster_name$-dm-worker-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}-dm-cluster-secret` Secret object to be used by the DM-worker component of the DM cluster.

- A set of client-side certificates of DM cluster components.

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

  `${cluster_name}` is the name of the cluster. The above YAML file creates three objects:

  - Set `spec.secretName` to `${cluster_name}-dm-master-cluster-secret`.
  - Add `server auth` and `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

  After the object is created, `cert-manager` generates a `${cluster_name}-cluster-client-secret` Secret object to be used by the clients of the DM components.

7.1.3.1.2 Deploy the DM cluster

When you deploy a DM cluster, you can enable TLS between DM components, and set the `cert-allowed-cn` configuration item 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 DM-master. Therefore, the `commonName` of all `Certificate` objects must be the same.

- Create the `dm-cluster.yaml` file:

```
apiVersion: pingcap.com/v1alpha1
kind: DMCluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  tlsCluster:
    enabled: true
  version: v8.5.0
  pvReclaimPolicy: Retain
  discovery: {}
  master:
    baseImage: pingcap/dm
    maxFailoverCount: 0
    replicas: 1
    storageSize: "1Gi"
    config:
      cert-allowed-cn:
        - TiDB
  worker:
    baseImage: pingcap/dm
    maxFailoverCount: 0
    replicas: 1
    storageSize: "1Gi"
    config:
      cert-allowed-cn:
        - TiDB
```

Use the `kubectl apply -f dm-cluster.yaml` file to create a DM cluster.

7.1.3.1.3 Configure `dmctl` and connect to the cluster

Get into the DM-master Pod:

```
kubectl exec -it ${cluster_name}-dm-master-0 -n ${namespace} sh
```

Use `dmctl`:

```
cd /var/lib/dm-master-tls
/dmctl --ssl-ca=ca.crt --ssl-cert=tls.crt --ssl-key=tls.key --master-addr
    ↪ 127.0.0.1:8261 list-member
```

7.1.3.2 Use DM to migrate data between MySQL/TiDB databases that enable TLS for the MySQL client

This section describes how to configure DM to migrate data between MySQL/TiDB databases that enable TLS for the MySQL client.

To learn how to enable TLS for the MySQL client of TiDB, refer to Enable TLS for the MySQL Client.

7.1.3.2.1 Step 1: Create the Kubernetes Secret object for each TLS-enabled MySQL

Suppose you have deployed a MySQL/TiDB database with TLS-enabled for the MySQL client. To create Secret objects for the TiDB cluster, execute the following command:

```
kubectl create secret generic ${mysql_secret_name1} --namespace=${namespace}
    ↪  --from-file=tls.crt=client.pem --from-file=tls.key=client-key.pem --
    ↪ from-file=ca.crt=ca.pem
kubectl create secret generic ${tidb_secret_name} --namespace=${namespace}
    ↪ --from-file=tls.crt=client.pem --from-file=tls.key=client-key.pem --
    ↪ from-file=ca.crt=ca.pem
```

7.1.3.2.2 Step 2: Mount the Secret objects to the DM cluster

After creating the Kubernetes Secret objects for the upstream and downstream databases, you need to set `spec.tlsClientSecretNames` so that you can mount the Secret objects to the Pod of DM-master/DM-worker.

```
apiVersion: pingcap.com/v1alpha1
kind: DMCluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  version: v8.5.0
  pvReclaimPolicy: Retain
```

```
discovery: {}
tlsClientSecretNames:
  - ${mysql_secret_name1}
  - ${tidb_secret_name}
master:
  ...
```

7.1.3.2.3 Step 3: Modify the data source and migration task configuration

After configuring `spec.tlsClientSecretNames`, TiDB Operator will mount the Secret objects `${secret_name}` to the path `/var/lib/source-tls/${secret_name}`.

1. Configure `from.security` in the `source1.yaml` file as described in the data source configuration:

```
source-id: mysql-replica-01
relay-dir: /var/lib/dm-worker/relay
from:
  host: ${mysql_host1}
  user: dm
  password: ""
  port: 3306
  security:
    ssl-ca: /var/lib/source-tls/${mysql_secret_name1}/ca.crt
    ssl-cert: /var/lib/source-tls/${mysql_secret_name1}/tls.crt
    ssl-key: /var/lib/source-tls/${mysql_secret_name1}/tls.key
```

2. Configure `target-database.security` in the `task.yaml` file as described in the Configure Migration Tasks:

```
name: test
task-mode: all
is-sharding: false

target-database:
  host: ${tidb_host}
  port: 4000
  user: "root"
  password: ""
  security:
    ssl-ca: /var/lib/source-tls/${tidb_secret_name}/ca.crt
    ssl-cert: /var/lib/source-tls/${tidb_secret_name}/tls.crt
    ssl-key: /var/lib/source-tls/${tidb_secret_name}/tls.key
```

```
mysql-instances:
- source-id: "replica-01"
  loader-config-name: "global"

loaders:
  global:
    dir: "/var/lib/dm-worker/dumped_data"
```

7.1.3.2.4 Step 4: Start the migration tasks

Refer to Start the migration tasks.

7.1.4 Replicate Data to TLS-enabled Downstream Services

This document describes how to replicate data to TLS-enabled downstream services on Kubernetes.

7.1.4.1 Preparations

Before you begin, do the following preparations:

1. Deploy a downstream service, and enable the TLS authentication on the client.
2. Generate the key file required for the client to access the downstream service.

7.1.4.2 Steps

1. Create a Kubernetes Secret object that contains a client TLS certificate used to access the downstream service. You can get the certificate from the key file you generated for the client.

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

2. Mount the certificate file to the TiCDC Pod.

    - If you have not deployed a TiDB cluster yet, add the `spec.ticdc.tlsClientSecretNames` ↪ field to the TidbCluster CR definition, and then deploy the TiDB cluster.
    - If you have already deployed a TiDB cluster, run `kubectl edit tc $` ↪ `{cluster_name} -n ${cluster_namespace}`, add the `spec.tiddc.` ↪ `tlsClientSecretNames` field, and then wait for the TiCDC pod to automatically roll over for updates.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${cluster_name}
  namespace: ${cluster_namespace}
spec:
  # ...
  ticdc:
    baseImage: pingcap/ticdc
    version: "v5.0.1"
    # ...
    tlsClientSecretNames:
    - ${secret_name}
```

Once the TiCDC Pod is running, the created Kubernetes Secret object is mounted to the TiCDC Pod. You can get the mounted key file in the `/var/lib/sink-tls/${secret_name}` directory of the Pod.

3. Create a replication task using the `cdc cli` tool.

```
kubectl exec ${cluster_name}-ticdc-0 -- /cdc cli changefeed create --pd
    ↪ =https://${cluster_name}-pd:2379 --sink-uri="mysql://${user}:{
    ↪ $password}@${downstream_service}/?ssl-ca=/var/lib/sink-tls/${
    ↪ secret_name}/ca.crt&ssl-cert=/var/lib/sink-tls/${secret_name}/tls
    ↪ .crt&ssl-key=/var/lib/sink-tls/${secret_name}/tls.key"
```

7.1.5 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.

7.1.5.1 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.

7.1.5.1.1 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, TiFlash and TiProxy, 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.

7.1.5.1.2 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, TiFlash and TiProxy, 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.

7.1.5.1.3 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.

7.1.5.2 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.

7.1.5.2.1 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.

7.1.5.2.2 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 regenerate 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.

7.1.6 Run Containers as a Non-root User

In some Kubernetes environments, containers cannot be run as the root user. In this case, you can set `securityContext` to run containers as a non-root user.

7.1.6.1 Configure TiDB Operator containers

For TiDB Operator containers, you can configure security context in the Helm `values.yaml` file. All TiDB Operator components (at `<controllerManager/scheduler/advancedStatefulset/admissionWebhook>.securityContext`) support this configuration.

The following is an example configuration:

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

7.1.6.2 Configure containers controlled by CR

For the containers controlled by Custom Resource (CR), you can configure security context in any CRs (`TidbCluster/DmCluster/TidbInitializer/TidbMonitor/Backup/BackupSchedule/Restore`) to make the containers run as a non-root user.

You can use either of the following two types of configuration. If you configure both the cluster level and the component level for a component, only the configuration of the component level takes effect.

- Configure `podSecurityContext` at the cluster level (`spec.podSecurityContext`) for all components. The following is an example configuration:

```
spec:
  podSecurityContext:
    runAsUser: 1000
    runAsGroup: 2000
    fsGroup: 2000
```

- Configure at the component level for a specific component. For example, configuring `spec.tidb.podSecurityContext` for `TidbCluster`, `spec.master.podSecurityContext` for `DMCluster`. The following is an example configuration:

```
spec:
  pd:
    podSecurityContext:
      runAsUser: 1000
      runAsGroup: 2000
      fsGroup: 2000
  tidb:
    podSecurityContext:
      runAsUser: 1000
      runAsGroup: 2000
      fsGroup: 2000
```

7.2 Manually Scale TiDB on Kubernetes

This document introduces how to horizontally and vertically scale a TiDB cluster on Kubernetes.

7.2.1 Horizontal scaling

Horizontally scaling TiDB means that you scale TiDB out or in by adding or remove Pods 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.

- To scale out a TiDB cluster, **increase** the value of `replicas` of a certain component. The scaling out operations add Pods based on the Pod ID in ascending order, until the number of Pods equals the value of `replicas`.
- To scale in a TiDB cluster, **decrease** the value of `replicas` of a certain component. The scaling in operations remove Pods based on the Pod ID in descending order, until the number of Pods equals the value of `replicas`.

7.2.1.1 Horizontally scale PD, TiKV, TiDB, and TiProxy

To scale PD, TiKV, TiDB, or TiProxy horizontally, use kubectl to modify `spec.pd.replicas`, `spec.tikv.replicas`, `spec.tidb.replicas`, and `spec.tiproxy.replicas` in the `TidbCluster` object of the cluster to desired values.

1. Modify the `replicas` value of a component as needed. For example, configure the `replicas` value of PD to 3:

```
kubectl patch -n ${namespace} tc ${cluster_name} --type merge --patch
    ↪ '{"spec":{"pd":{"replicas":3}}}'
```

2. Check whether your configuration has been updated in the corresponding TiDB cluster on Kubernetes.

```
kubectl get tidbcluster ${cluster_name} -n ${namespace} -oyaml
```

   If your configuration is successfully updated, in the `TidbCluster` CR output by the command above, the values of `spec.pd.replicas`, `spec.tidb.replicas`, and `spec.tikv.replicas` are consistent with the values you have configured.

3. Check whether the number of `TidbCluster` Pods has increased or decreased.

```
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.

7.2.1.2 Horizontally scale TiFlash

This section describes how to horizontally scale out or scale in TiFlash if you have deployed TiFlash in the cluster.

7.2.1.2.1 Horizontally scale out TiFlash

To scale out TiFlash horizontally, you can modify `spec.tiflash.replicas`.

For example, configure the `replicas` value of TiFlash to 3:

```
kubectl patch -n ${namespace} tc ${cluster_name} --type merge --patch '{"
    ↪ spec":{"tiflash":{"replicas":3}}}'
```

7.2.1.2.2 Horizontally scale in TiFlash

To scale in TiFlash horizontally, perform the following steps:

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 ${
    ↪ pod_number};
```

      `${pod_number}` indicates the number of remaining Pods in the TiFlash cluster after scaling in.

5. Wait for the number of TiFlash replicas in the related tables to be updated.

   Connect to the TiDB service, and run the following command to check the number:

```
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 on 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
```

7.2.1.3 Horizontally scale TiCDC

If TiCDC is deployed in the cluster, you can horizontally scale out or scale in TiCDC by modifying the value of `spec.ticdc.replicas`.

For example, configure the `replicas` value of TiCDC to 3:

```
kubectl patch -n ${namespace} tc ${cluster_name} --type merge --patch '{"
    ↪ spec":{"ticdc":{"replicas":3}}}'
```

7.2.1.4 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.

7.2.2 Vertical scaling

Vertically scaling TiDB means that you scale TiDB up or down by increasing or decreasing the limit of resources on the Pod. Vertical scaling is essentially the rolling update of the Pods.

7.2.2.1 Vertically scale components

This section describes how to vertically scale up or scale down components including PD, TiKV, TiDB, TiProxy, TiFlash, and TiCDC.

- To scale up or scale down PD, TiKV, TiDB, and TiProxy, use kubectl to modify `spec.pd.resources`, `spec.tikv.resources`, `spec.tidb.resources`, and `spec.tiproxy.replicas` in the `TidbCluster` object that corresponds to the cluster to desired values.
- To scale up or scale down TiFlash, modify the value of `spec.tiflash.resources`.
- To scale up or scale down TiCDC, modify the value of `spec.ticdc.resources`.

7.2.2.2 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.

7.2.3 Scale PD microservice components

Note:

Starting from v8.0.0, PD supports the microservice mode (experimental).

PD microservices are typically used to address performance bottlenecks in PD and improve the quality of PD services. To determine whether it is necessary to scale PD microservices, see PD microservice FAQs.

- Currently, the PD microservices mode splits the timestamp allocation and cluster scheduling functions of PD into two independently deployed components: the `tso` microservice and the `scheduling` microservice.
  - The `tso` microservice implements a primary-secondary architecture. If the `tso` microservice becomes the bottleneck, it is recommended to scale it vertically.
  - The `scheduling` microservice serves as a scheduling component. If the `scheduling` microservice becomes the bottleneck, it is recommended to scale it horizontally.
- To vertically scale each component of PD microservices, use the `kubectl` command to modify the `spec.pdms.resources` of the `TidbCluster` object corresponding to the cluster to your desired value.
- To horizontally scale each component of PD microservices, use the `kubectl` command to modify `spec.pdms.replicas` of the `TidbCluster` object corresponding to the cluster to your desired value.

Taking the `scheduling` microservice as an example, the steps for horizontal scaling are as follows:

1. Modify the `replicas` value of the corresponding `TidbCluster` object to your desired value. For example, run the following command to set the `replicas` value of `scheduling` to 3:

```
kubectl patch -n ${namespace} tc ${cluster_name} --type merge --patch
    ↪ '{"spec":{"pdms":[{"name":"scheduling", "replicas":3}]}}'
```

2. Check whether the corresponding TiDB cluster configuration for the Kubernetes cluster is updated:

```
kubectl get tidbcluster ${cluster_name} -n ${namespace} -oyaml
```

   In the output of this command, the `scheduling.replicas` value of `spec.pdms` in `TidbCluster` is expected to be the same as the value you configured.

3. Observe whether the number of `TidbCluster` Pods is increased or decreased:

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

It usually takes about 10 to 30 seconds for PD microservice components to scale in or out.

7.2.4 Scaling troubleshooting

During the horizontal or vertical scaling operation, Pods might go to the Pending state because of insufficient resources. See Troubleshoot the Pod in Pending state to resolve it.

7.3 Upgrade

7.3.1 Upgrade a TiDB Cluster on Kubernetes

If you deploy and manage your TiDB clusters on Kubernetes using TiDB Operator, you can upgrade your TiDB clusters using the rolling update feature. Rolling update can limit the impact of upgrade on your application.

This document describes how to upgrade a TiDB cluster on Kubernetes using rolling updates.

7.3.1.1 Rolling update introduction

Kubernetes provides the rolling update feature to update your application with zero downtime.

When you perform a rolling update, TiDB Operator serially deletes an old Pod and creates the corresponding new Pod in the order of PD, TiProxy, TiFlash, TiKV, and TiDB. After the new Pod runs normally, TiDB Operator proceeds with the next Pod.

> **Note:**
>
> If PD microservices (introduced in TiDB v8.0.0) are deployed in a cluster, when you perform a rolling update to upgrade the cluster, TiDB Operator serially deletes an old Pod and creates the corresponding new Pod in the order of each PD microservice component, PD, TiKV, and TiDB. After the new Pod runs normally, TiDB Operator proceeds with the next Pod.

During the rolling update, TiDB Operator automatically completes Leader transfer for PD and TiKV. 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 application. If your client supports retrying stale connections, performing a rolling update to TiDB servers does not impact application, either.

Warning:

- For the clients that cannot retry stale connections, **performing a rolling update to TiDB servers closes the client connections and cause the request to fail**. In such cases, it is recommended to add a retry function for the clients to retry, or to perform a rolling update to TiDB servers in idle time.
- Before upgrading, refer to the documentation to confirm that there are no DDL operations in progress.

7.3.1.2 Preparations before upgrade

1. Refer to the upgrade caveat to learn about the precautions. Note that all TiDB versions, including patch versions, currently do not support downgrade or rollback after upgrade.

2. Refer to TiDB release notes to learn about the compatibility changes in each intermediate version. If any changes affect your upgrade, take appropriate measures.

   For example, if you upgrade from TiDB v6.4.0 to v6.5.2, you need to check the compatibility changes in the following versions:

   - TiDB v6.5.0 compatibility changes and deprecated features
   - TiDB v6.5.1 compatibility changes
   - TiDB v6.5.2 compatibility changes

   If you upgrade from v6.3.0 or an earlier version to v6.5.2, you also need to check the compatibility changes in all intermediate versions.

7.3.1.3 Upgrade steps

Note:

By default, TiDB (versions starting from v4.0.2 and released before February 20, 2023) 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. Starting from February 20, 2023, the telemetry feature is disabled by default in newly released TiDB versions. See TiDB Release Timeline for details.

1. In `TidbCluster` CR, modify the image configurations of all components of the cluster to be upgraded.

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

   Usually, all 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, modify `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`.

   The `version` field has following formats:

   - `spec.version`: the format is `imageTag`, such as `v8.5.0`
   - `spec.<pd/tidb/tikv/pump/tiflash/ticdc>.version`: the format is `imageTag`, such as `v3.1.0`

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.

7.3.1.4 Troubleshoot the upgrade

If the PD cluster is unavailable due to PD configuration errors, PD image tag errors, NodeAffinity, or other causes, you might not be able to successfully upgrade the TiDB cluster. In such cases, you can force an upgrade of the cluster to recover the cluster functionality.

The steps of force upgrade are as follows:

1. Set `annotation` for the cluster:

```
kubectl annotate --overwrite tc ${cluster_name} -n ${namespace} tidb.
    ↪ pingcap.com/force-upgrade=true
```

2. Change the related PD configuration to make sure that PD turns into a normal state.

3. After the PD cluster recovers, you *must* execute the following command to disable the forced upgrade; otherwise, an exception may occur in the next upgrade:

```
kubectl annotate tc ${cluster_name} -n ${namespace} tidb.pingcap.com/
    ↪ force-upgrade-
```

After taking the steps above, your TiDB cluster recovers its functionality. You can upgrade the cluster normally.

7.3.2 Upgrade TiDB Operator

7.3.2.1 Upgrade TiDB Operator

This document describes how to upgrade TiDB Operator to a specific version. You can choose either online upgrade or offline upgrade.

7.3.2.1.1 Online upgrade

If your server has access to the internet, you can perform online upgrade by taking the following steps:

1. Before upgrading TiDB Operator, make sure that the Helm repo contains the TiDB Operator version you want to upgrade to. To check the TiDB Operator versions in the Helm repo, run the following command:

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

   If the command output does not include the version you need, update the repo using the `helm repo update` command. For details, refer to Configure the Help repo.

2. Update CustomResourceDefinition (CRD) for Kubernetes:

   1. If you upgrade TiDB Operator from v1.3.x to v1.4.0 or later versions, you need to execute the following command to create the new TidbDashboard CRD. If you upgrade TiDB Operator from v1.4.0 or later versions, you can skip this step.

```
kubectl create -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/${operator_version}/manifests/crd/v1/pingcap.
    ↪ com_tidbdashboards.yaml
```

   2. Update CRD.

```
kubectl replace -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/${operator_version}/manifests/crd.yaml && \
kubectl get crd tidbclusters.pingcap.com
```

   This document takes TiDB v1.6.1 as an example. You can replace `${operator_version ↪ }` with the specific version you want to upgrade to.

3. Get the `values.yaml` file of the `tidb-operator` chart:

```
mkdir -p ${HOME}/tidb-operator/v1.6.1 && \
helm inspect values pingcap/tidb-operator --version=v1.6.1 > ${HOME}/
    ↪ tidb-operator/v1.6.1/values-tidb-operator.yaml
```

4. In the `${HOME}/tidb-operator/v1.6.1/values-tidb-operator.yaml` file, modify the `operatorImage` version to the new TiDB Operator version.

5. If you have added customized configuration in the old `values.yaml` file, merge your customized configuration to the `${HOME}/tidb-operator/v1.6.1/values-tidb-operator.yaml` file.

6. Perform upgrade:

```
helm upgrade tidb-operator pingcap/tidb-operator --version=v1.6.1 -f ${HOME}/tidb-operator/v1.6.1/values-tidb-operator.yaml -n tidb-admin
```

7. After all the Pods start normally, 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 you see a similar output as follows, TiDB Operator is successfully upgraded. `v1.6.1` represents the TiDB Operator version you have upgraded to.

```
image: pingcap/tidb-operator:v1.6.1
image: docker.io/pingcap/tidb-operator:v1.6.1
image: pingcap/tidb-operator:v1.6.1
image: docker.io/pingcap/tidb-operator:v1.6.1
```

7.3.2.1.2 Offline upgrade

If your server cannot access the Internet, you can offline upgrade by taking the following steps:

1. Download the files and images required for the upgrade using a machine with Internet access:

   1. Download the `crd.yaml` file for the new TiDB Operator version. For more information about CRD, see CustomResourceDefinition.

```
wget -O crd.yaml https://raw.githubusercontent.com/pingcap/tidb-operator/${operator_version}/manifests/crd.yaml
```

      This document takes TiDB v1.6.1 as an example. You can replace `${operator_version}` with the specific version you want to upgrade to.

   2. Download the `tidb-operator` chart package file.

```
wget http://charts.pingcap.org/tidb-operator-v1.6.1.tgz
```

   3. Download the Docker images required for the new TiDB Operator version:

```
docker pull pingcap/tidb-operator:v1.6.1
docker pull pingcap/tidb-backup-manager:v1.6.1

docker save -o tidb-operator-v1.6.1.tar pingcap/tidb-operator:v1
    ↪ .6.1
docker save -o tidb-backup-manager-v1.6.1.tar pingcap/tidb-backup-
    ↪ manager:v1.6.1
```

2. Upload the downloaded files and images to the server where TiDB Operator is deployed, and install the new TiDB Operator version:

    1. If you upgrade TiDB Operator from v1.2.x or earlier versions to v1.3.x or later versions, you need to execute the following command to create the new TidbNGMonitoring CRD. If you upgrade TiDB Operator from v1.3.x or later versions, you can skip this step.

        ```
        kubectl create -f ./crd.yaml
        ```

        After executing this command, you can expect to see an "AlreadyExists" error for other CRDs. You can ignore this error.

    2. Install the `crd.yaml` file for TiDB Operator:

        ```
        kubectl replace -f ./crd.yaml
        ```

    3. Unpack the `tidb-operator` chart package file, and copy the `values.yaml` file to the directory of the new TiDB Operator:

        ```
        tar zxvf tidb-operator-v1.6.1.tgz && \
        mkdir -p ${HOME}/tidb-operator/v1.6.1 && \
        cp tidb-operator/values.yaml ${HOME}/tidb-operator/v1.6.1/values-
            ↪ tidb-operator.yaml
        ```

    4. Install the Docker images on the server:

        ```
        docker load -i tidb-operator-v1.6.1.tar && \
        docker load -i tidb-backup-manager-v1.6.1.tar
        ```

3. In the `${HOME}/tidb-operator/v1.6.1/values-tidb-operator.yaml` file, modify the `operatorImage` version to the new TiDB Operator version.

4. If you have added customized configuration in the old `values.yaml` file, merge your customized configuration to the `${HOME}/tidb-operator/v1.6.1/values-tidb-operator.yaml` file.

5. Perform upgrade:

```
helm upgrade tidb-operator ./tidb-operator --version=v1.6.1 -f ${HOME}/
    ↪ tidb-operator/v1.6.1/values-tidb-operator.yaml
```

6. After all the Pods start normally, check the image version of TiDB Operator:

```
kubectl get po -n tidb-admin -l app.kubernetes.io/instance=tidb-
    ↪ operator -o yaml | grep 'image:.*operator:'
```

If you see a similar output as follows, TiDB Operator is successfully upgraded. `v1.6.1` represents the TiDB Operator version you have upgraded to.

```
image: pingcap/tidb-operator:v1.6.1
image: docker.io/pingcap/tidb-operator:v1.6.1
image: pingcap/tidb-operator:v1.6.1
image: docker.io/pingcap/tidb-operator:v1.6.1
```

> **Note:**
> After TiDB Operator is upgraded, the `discovery` Deployment in all TiDB clusters is automatically upgraded to the corresponding version of TiDB Operator.

7.3.2.2 Perform a Canary Upgrade on TiDB Operator

If you want to upgrade TiDB Operator to a new version, and hope to limit the impact of the upgrade to avoid the unpredictable impact on all TiDB clusters in the entire Kubernetes cluster, you can perform a canary upgrade on TiDB Operator. After the canary upgrade, you can check the impact of the TiDB Operator upgrade on the canary cluster. After you confirm that the new version of TiDB Operator is working stably, you can then upgrade TiDB Operator normally.

You can perform a canary upgrade only on two components: `tidb-controller-manager` ↪ and `tidb-scheduler`. Canary upgrades for the advanced StatefulSet controller and the admission controller are not supported.

When you use TiDB Operator, `tidb-scheduler` is not mandatory. Refer to tidb-scheduler and default-scheduler to confirm whether you need to deploy `tidb-scheduler`.

7.3.2.2.1 Step 1: Configure selector for the current TiDB Operator and perform an upgrade

In `values.yaml` of the current TiDB Operator, add the following selector configuration:

```
controllerManager:
  selector:
  - version!=canary
```

Refer to Online upgrade or Offline upgrade to upgrade the current TiDB Operator:

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

7.3.2.2.2 Step 2: Deploy the canary TiDB Operator

1. Refer to Step 1~2 in Online deployment and obtain the `values.yaml` file of the version you want to upgrade to. Add the following configuration in `values.yaml`:

   ```
   controllerManager:
     selector:
     - version=canary
   appendReleaseSuffix: true
   #scheduler:
   #  create: false # If you do not need tidb-scheduler, set this value to
       ↪  false.
   advancedStatefulset:
     create: false
   admissionWebhook:
     create: false
   ```

   `appendReleaseSuffix` must be set to `true`.

   If you do not need to perform a canary upgrade on `tidb-scheduler`, configure `scheduler.create: false`. If you need to perform a canary upgrade on `tidb-scheduler`, configuring `scheduler.create: true` creates a scheduler named `{{ .scheduler.schedulerName }}-{{.Release.Name}}`. To use this scheduler in the canary TiDB Operator, in the `TidbCluster` CR, configure `spec.schedulerName` to the name of this scheduler.

   Because canary upgrades for the advanced StatefulSet controller and the admission controller are not supported, you need to set `advancedStatefulset.create: false` and `admissionWebhook.create: false`.

   For details on the parameters related to canary upgrade, refer to related parameters.

2. 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 ...`):

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

   Replace `${operator_version}` with the version of TiDB Operator you want to upgrade to.

7.3.2.2.3 Step 3: Test the canary TiDB Operator (optional)

Before you upgrade TiDB Operator in a normal way, you can test whether the canary TiDB Operator works stably. You can test `tidb-controller-manager` and `tidb-scheduler` ↪ .

1. To test the canary `tidb-controller-manager`, set a label 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 with the `canary` label is now managed by the canary TiDB Operator. The steps to check logs are as follows:

   1. View the log of `tidb-controller-manager` of the current TiDB Operator:

```
kubectl -n tidb-admin logs tidb-controller-manager-55b887bdc9-lzdwv
```

      Expected output:

```
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
```

      Expected output:

```
I0113 03:38:43.859387    1 tidbcluster_control.go:69] TidbCluster:
   ↪ [tidb-cluster-1/basic1] updated successfully
```

2. To test the canary upgrade of `tidb-scheduler`, modify `spec.schedulerName` of a 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
```

3. After the tests, you can revert the changes in the previous two steps 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}
```

7.3.2.2.4 Step 4: Upgrade TiDB Operator normally

After you confirm that the canary TiDB Operator works stably, you can upgrade the TiDB Operator normally.

1. Delete the canary TiDB Operator:

```
helm -n tidb-admin-canary uninstall ${release_name}
```

2. Upgrade TiDB Operator normally.

7.4 Backup and Restore

7.4.1 Backup and Restore Overview

This document describes how to perform backup and restore on the TiDB cluster on Kubernetes. To back up and restore your data, you can use the Dumpling, TiDB Lightning, and Backup & Restore (BR) tools.

Dumpling is a data export tool, which exports data stored in TiDB or MySQL as SQL or CSV data files. You can use Dumpling to make a logical full backup or export.

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 to make a logical full data restore or import.

BR is a command-line tool for distributed backup and restoration of the TiDB cluster data. Compared with Dumpling and Mydumper, BR is more suitable for huge data volumes. BR only supports TiDB v3.1 and later versions. For incremental backup insensitive to latency, refer to BR Overview. For real-time incremental backup, refer to TiCDC.

7.4.1.1 Usage scenarios

7.4.1.1.1 Back up data

If you have the following backup needs, you can use BR to make a backup of your TiDB cluster data:

- To back up a large volume of data (more than 1 TB) at a fast speed
- To get a direct backup of data as SST files (key-value pairs)
- To perform incremental backup that is insensitive to latency

Refer to the following documents for more information:

- Back up Data to S3-Compatible Storage Using BR
- Back up Data to GCS Using BR

- Back up Data to Azure Blob Storage Using BR
- Back up Data to PV Using BR
- Back up Data Using EBS Snapshots across Multiple Kubernetes

If you have the following backup needs, you can use Dumpling to make a backup of the TiDB cluster data:

- To export SQL or CSV files
- To limit the memory usage of a single SQL statement
- To export the historical data snapshot of TiDB

Refer to the following documents for more information:

- Back up Data to S3-Compatible Storage Using Dumpling
- Back up Data to GCS Using Dumpling

7.4.1.1.2 Restore data

To recover the SST files exported by BR to a TiDB cluster, use BR. Refer to the following documents for more information:

- Restore Data from S3-Compatible Storage Using BR
- Restore Data from GCS Using BR
- Restore Data from Azure Blob Storage Using BR
- Restore Data from PV Using BR
- Restore Data Using EBS Snapshots across Multiple Kubernetes

To restore data from SQL or CSV files exported by Dumpling or other compatible data sources to a TiDB cluster, use TiDB Lightning. Refer to the following documents for more information:

- Restore Data from S3-Compatible Storage Using TiDB Lightning
- Restore Data from GCS Using TiDB Lightning

7.4.1.2 Backup and restore process

To make a backup of the TiDB cluster on Kubernetes, you need to create a `Backup` CR object to describe the backup or create a `BackupSchedule` CR object to describe a scheduled backup.

To restore data to the TiDB cluster on Kubernetes, you need to create a `Restore` CR object to describe the restore.

After creating the CR object, according to your configuration, TiDB Operator chooses the corresponding tool and performs the backup or restore.

7.4.1.3 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 cleans the backup data when it deletes the CR.

If you use v1.5.5, v1.6.1, or a later version, TiDB Operator automatically attempts to stop running log backup tasks when you delete the Custom Resource (CR). This automatic stop feature only applies to log backup tasks that are running normally and does not handle tasks in an error or failed state.

If you back up cluster data using AWS EBS volume snapshots and set the value of `spec.cleanPolicy` to `Delete`, TiDB Operator deletes the CR, and cleans up the backup file and the volume snapshots on AWS.

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

7.4.1.3.1 Clean backup data

For TiDB Operator v1.2.3 and earlier versions, TiDB Operator cleans the backup data by deleting the backup files one by one.

For TiDB Operator v1.2.4 and later versions, TiDB Operator cleans the backup data by deleting the backup files in batches. For the batch deletion, the deletion methods are different depending on the type of backend storage used for backups.

- For the S3-compatible backend storage, TiDB Operator uses the concurrent batch deletion method, which deletes files in batch concurrently. TiDB Operator starts multiple goroutines concurrently, and each goroutine uses the batch delete API "DeleteObjects" to delete multiple files.
- For other types of backend storage, TiDB Operator uses the concurrent deletion method, which deletes files concurrently. TiDB Operator starts multiple goroutines, and each goroutine deletes one file at a time.

For TiDB Operator v1.2.4 and later versions, you can configure the following fields in the Backup CR to control the clean behavior:

- `.spec.cleanOption.pageSize`: Specifies the number of files to be deleted in each batch at a time. The default value is 10000.
- `.spec.cleanOption.disableBatchConcurrency`: If the value of this field is `true`, TiDB Operator disables the concurrent batch deletion method and uses the concurrent deletion method.

  If your S3-compatible backend storage does not support the `DeleteObjects` API, the default concurrent batch deletion method fails. You need to configure this field to `true` to use the concurrent deletion method.
- `.spec.cleanOption.batchConcurrency`: Specifies the number of goroutines to start for the concurrent batch deletion method. The default value is `10`.
- `.spec.cleanOption.routineConcurrency`: Specifies the number of goroutines to start for the concurrent deletion method. The default value is `100`.

7.4.2 Backup and Restore Custom Resources

This document describes the fields in the `Backup`, `Restore`, and `BackupSchedule` custom resources (CR). You can use these fields to better perform the backup or restore of TiDB clusters on Kubernetes.

7.4.2.1 Backup CR fields

To back up data for a TiDB cluster on Kubernetes, you can create a `Backup` custom resource (CR) object. For detailed backup process, refer to documents listed in Back up data.

This section introduces the fields in the `Backup` CR.

7.4.2.1.1 General fields

- `.spec.metadata.namespace`: the namespace where the `Backup` CR is located.
- `.spec.toolImage`: the tool image used by `Backup`. TiDB Operator supports this configuration item starting from v1.1.9.
  - 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:v8.5.0`, the image of the specified version is used for backup.

    * If an image is specified without the version, 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.
    * If the Dumpling version is specified in this field, such as `spec.toolImage: pingcap/dumpling:v8.5.0`, the image of the specified version is used for backup.
    * If the field is not specified, the Dumpling version specified in `TOOLKIT_VERSION` of the Backup Manager Dockerfile is used for backup by default.

- `.spec.backupType`: the backup type. This field is valid only when you use BR for backup. Currently, the following three types are supported, and this field can be combined with the `.spec.tableFilter` field to configure table filter rules:
  - `full`: back up all databases in a TiDB cluster.
  - `db`: back up a specified database in a TiDB cluster.
  - `table`: back up a specified table in a TiDB cluster.

- `.spec.backupMode`: the backup mode. The default value is `snapshot`, which means backing up data through the snapshots in the KV layer. This field is valid only for backup and has three value options currently:
  - `snapshot`: back up data through snapshots in the KV layer.
  - `volume-snapshot`: back up data by volume snapshots.
  - `log`: back up log data in real time in the KV layer.

- `.spec.logSubcommand`: the subcommand for controlling the log backup status in the Backup CR. This field provides three options for managing a log backup task:
  - `log-start`: initiates a new log backup task or resumes a paused task. Use this command to start the log backup process or resume a task from a paused state.
  - `log-pause`: temporarily pauses the currently running log backup task. After pausing, you can use the `log-start` command to resume the task.
  - `log-stop`: permanently stops the log backup task. After executing this command, the Backup CR enters a stopped state and cannot be restarted.

For versions before v1.5.5, use the `logStop` field with boolean values (`true`/`false`) to control log backup operations. While `logStop` is still supported in v1.5.5 and v1.6.1, it is recommended to use `logSubcommand` instead.

- `.spec.restoreMode`: the restore mode. The default value is `snapshot`, which means restoring data from snapshots in the KV layer. This field is valid only for restore and has three value options currently:

  - `snapshot`: restore data from snapshots in the KV layer.
  - `volume-snapshot`: restore data from volume snapshots.
  - `pitr`: restore cluster data to a specific point in time based on snapshots and log data.

- `.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, whether the backup is successful or not, as long as the previous `tikv_gc_life_time` value is smaller than `.spec.tikvGCLifeTime`, TiDB Operator tries 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.

  In such cases, 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 manually set `tikv_gc_life_time` back to the previous value.

- `.spec.cleanPolicy`: The cleaning policy for the backup data when the backup CR is deleted. You can choose one from the following three clean policies:

  - `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.cleanOption`: the clean behavior for the backup files when the backup CR is deleted after the cluster backup. For details, refer to Clean backup data

- `.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 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` created when you 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 GiB` 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 first delete this PVC and then run the Backup job.
- `.spec.resources`: the resource requests and limits for the Pod that runs the backup job.
- `.spec.env`: the environment variables for the Pod that runs the backup job.
- `.spec.affinity`: the affinity configuration for the Pod that runs the backup job. For details on affinity, refer to Affinity and anti-affinity.
- `.spec.tolerations`: specifies that the Pod that runs the backup job can schedule onto nodes with matching taints. For details on taints and tolerations, refer to Taints and Tolerations.
- `.spec.podSecurityContext`: the security context configuration for the Pod that runs the backup job, which allows the Pod to run as a non-root user. For details on `podSecurityContext`, refer to Run Containers as a Non-root User.

- `.spec.priorityClassName`: the name of the priority class for the Pod that runs the backup job, which sets priority for the Pod. For details on priority classes, refer to Pod Priority and Preemption.
- `.spec.imagePullSecrets`: the imagePullSecrets for the Pod that runs the backup job.
- `.spec.serviceAccount`: the name of the ServiceAccount used for the backup.
- `.spec.useKMS`: whether to use AWS-KMS to decrypt the S3 storage key used for the backup.
- `.spec.tableFilter`: specifies tables that match the table filter rules for BR or Dumpling. This field can be ignored by default.

  When the field is not configured, if you use Dumpling, the default value of `tableFilter` is as follows:

```
tableFilter:
- "*.*"
- "!/^(mysql|test|INFORMATION_SCHEMA|PERFORMANCE_SCHEMA|METRICS_SCHEMA|
    ↪ INSPECTION_SCHEMA)$/.*"
```

  When the field is not configured, if you use BR, BR backs up all schemas except the system schema.

> **Note:**
>
> If you want to back up all tables except `db.table` using the `"!db.table"` rule, you need to first add the `*.*` rule to include all tables. For example:
>
> ```
> tableFilter:
> - "*.*"
> - "!db.table"
> ```

- `.spec.backoffRetryPolicy`: the retry policy for abnormal failures (such as Kubernetes killing the node due to insufficient resources) of the Job/Pod during the backup. This configuration currently only takes effect on the `snapshot` backup.
  - `minRetryDuration`: the minimum retry interval after an abnormal failure is found. The retry interval increases with the number of failures. `RetryDuration = minRetryDuration << (retryNum -1)`. The time format is specified in `func ParseDuration`, and the default value is `300s`.
  - `maxRetryTimes`: the maximum number of retries. The default value is `2`.
  - `retryTimeout`: the retry timeout. The timeout starts from the first abnormal failure. The time format is specified in `func ParseDuration`, and the default value is `30m`.

7.4.2.1.2 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, Google Cloud, or Azure permissions to the TiKV process. Defaults to `true`.
- `.spec.br.onLine`: whether to enable the online restore feature when restoring data.
- `.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.

7.4.2.1.3 S3 storage fields

- `.spec.s3.provider`: the supported S3-compatible storage provider. All supported providers 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 of the S3-compatible 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.path`: specifies the storage path of backup files on the remote storage. This field is valid only when the data is backed up using Dumpling or restored using TiDB Lightning. For example, `s3://test1-demo1/backup-2019-12-11T04:32:12Z.tgz`.
- `.spec.s3.endpoint:` the endpoint of S3 compatible storage service, for example, `http ↪ ://minio.minio.svc.cluster.local:9000`.
- `.spec.s3.secretName:` the name of secret which stores S3 compatible storage's access key and secret key.
- `.spec.s3.sse`: specifies the S3 server-side encryption method. For example, `aws:kms`.
- `.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.

7.4.2.1.4 GCS fields

- `.spec.gcs.projectId`: the unique identifier of the user project on Google Cloud. To obtain the project ID, refer to Google Cloud documentation.
- `.spec.gcs.location`: the location of the GCS bucket. For example, `us-west2`.

- `.spec.gcs.path`: the storage path of backup files on the remote storage. This field is valid only when the data is backed up using Dumpling or restored using TiDB Dumpling. For example, `gcs://test1-demo1/backup-2019-11-11T16:06:05Z.tgz`.
- `.spec.gcs.secretName`: the name of the secret that stores the GCS account credential.
- `.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`.
- `.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.

7.4.2.1.5 Azure Blob Storage fields

- `.spec.azblob.secretName`: the name of the secret which stores Azure Blob Storage account credential.
- `.spec.azblob.container`: the name of the container which stores data.
- `.spec.azblob.prefix`: if you set this field, the value is used to make up the remote storage path `azure://${.spec.azblob.container}/${.spec.azblob.prefix}/backupName`.
- `.spec.azblob.accessTier`: the access tier of the uploaded data.

  Azure Blob Storage supports the following access tier options:

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

  If this field is not configured, `Cool` is used by default.

7.4.2.1.6 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.

7.4.2.2 Restore CR fields

To restore data to a TiDB cluster on Kubernetes, you can create a `Restore` CR object. For detailed restore process, refer to documents listed in Restore data.

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`. TiDB Operator supports this configuration starting from v1.1.9.

  - When using BR for restoring, you can specify the BR version in this field. For example,`spec.toolImage: pingcap/br:v8.5.0`. 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:v8.5.0`. If not specified, the Lightning version specified in `TOOLKIT_VERSION` of the Backup Manager Dockerfile is used for restoring by default.

- `.spec.backupType`: the restore type. This field is valid only when you use BR to restore data. Currently, the following three types are supported, and this field can be combined with the `.spec.tableFilter` field to configure table filter rules:

  - `full`: restore all databases in a TiDB cluster.
  - `db`: restore a specified database in a TiDB cluster.
  - `table`: restore a specified table in a TiDB cluster.

- `.spec.tikvGCLifeTime`: the temporary `tikv_gc_life_time` setting during the restore, which defaults to `72h`.

  Before the restore begins, if the `tikv_gc_life_time` setting in the TiDB cluster is smaller than `spec.tikvGCLifeTime` set by users, TiDB Operator adjusts the value of `tikv_gc_life_time` to the value of `spec.tikvGCLifeTime`. This operation makes sure that the restored data is not garbage-collected by TiKV.

  After the restore, whether the restore is successful or not, as long as the original `tikv_gc_life_time` value is smaller than `.spec.tikvGCLifeTime`, TiDB Operator tries to set `tikv_gc_life_time` back to the original 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 restore as failed.

  In such cases, 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 manually set `tikv_gc_life_time` back to the previous value.

- `.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 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` created when you 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.resources`: the resource requests and limits for the Pod that runs the restore job.
- `.spec.env`: the environment variables for the Pod that runs the restore job.
- `.spec.affinity`: the affinity configuration for the Pod that runs the restore job. For details on affinity, refer to Affinity and anti-affinity.
- `.spec.tolerations`: specifies that the Pod that runs the restore job can schedule onto nodes with matching taints. For details on taints and tolerations, refer to Taints and Tolerations.
- `.spec.podSecurityContext`: the security context configuration for the Pod that runs the restore job, which allows the Pod to run as a non-root user. For details on `podSecurityContext`, refer to Run Containers as a Non-root User.
- `.spec.priorityClassName`: the name of the priority class for the Pod that runs the restore job, which sets priority for the Pod. For details on priority classes, refer to Pod Priority and Preemption.
- `.spec.imagePullSecrets`: the imagePullSecrets for the Pod that runs the restore job.
- `.spec.serviceAccount`: the name of the ServiceAccount used for restore.
- `.spec.useKMS`: whether to use AWS-KMS to decrypt the S3 storage key used for the backup.
- `.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.

  When the field is not configured, if you use TiDB Lightning, the default `tableFilter` value for TiDB Lightning is as follows:

```
tableFilter:
- "*.*"
- "!/^(mysql|test|INFORMATION_SCHEMA|PERFORMANCE_SCHEMA|METRICS_SCHEMA|
    ↪ INSPECTION_SCHEMA)$/.*"
```

When the field is not configured, if you use BR, BR restores all the schemas in the backup file.

> **Note:**
>
> If you want to backup up all table except `db.table` using the `"!db.table"` rule, 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.azblob`: Azure Blob Storage-related configuration. Refer to Azure Blob Storage fields.
- `.spec.local`: persistent volume-related configuration. Refer to Local storage fields.

7.4.2.3 BackupSchedule CR fields

The `backupSchedule` configuration consists of three parts: the configuration of the snapshot backup `backupTemplate`, the configuration of the log backup `logBackupTemplate`, and the unique configuration of `backupSchedule`.

- `backupTemplate`: the configuration of the snapshot backup. Specifies the configuration related to the cluster and remote storage of the snapshot backup, which is the same as the `spec` configuration of the `Backup` CR.
- `logBackupTemplate`: the configuration of the log backup. Specifies the configuration related to the cluster and remote storage of the log backup, which is the same as the `spec` configuration of the `Backup` CR. The log backup is created and deleted along with `backupSchedule` and recycled according to `.spec.maxReservedTime`. The log backup name is saved in `status.logBackup`.

> **Note:**
>
> Before you delete the log backup data, you need to stop the log backup task to avoid resource waste or the inability to restart the log backup task in the future because the log backup task in TiKV is not stopped.

- 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 point is reached. During this pause, the backup garbage collection runs normally. If you change `true` to `false`, the scheduled snapshot backup process is restarted. Because currently, log backup does not support pause, this configuration does not take effect for log backup.

7.4.3 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.

7.4.3.1 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.

7.4.3.1.1 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
```

7.4.3.1.2 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.

   When backing up a TiDB cluster using EBS volume snapshots, besides the `AmazonS3FullAccess` permission, the following permissions are also required:

```
{
"Effect": "Allow",
"Action": [
    "ec2:AttachVolume",
    "ec2:CreateSnapshot",
    "ec2:CreateSnapshots",
    "ec2:CreateTags",
    "ec2:CreateVolume",
    "ec2:DeleteSnapshot",
    "ec2:DeleteTags",
    "ec2:DeleteVolume",
    "ec2:DescribeInstances",
    "ec2:DescribeSnapshots",
    "ec2:DescribeTags",
    "ec2:DescribeVolumes",
    "ec2:DetachVolume",
    "ebs:ListSnapshotBlocks",
    "ebs:ListChangedBlocks"
```

```
        ],
        "Resource": "*"
    }
```

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 patch tc demo1 -n test1 --type merge -p '{"spec":{"tikv":{"
    ↪ annotations":{"iam.amazonaws.com/role":"arn:aws:iam
    ↪ ::123456789012:role/user"}}}}'
```

   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.

7.4.3.1.3 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` to grant tidb-backup-manager the access to this IAM role.

   When backing up a TiDB cluster using EBS volume snapshots, besides the `AmazonS3FullAccess` permission, the following permissions are also required:

```
        {
        "Effect": "Allow",
        "Action": [
            "ec2:AttachVolume",
```

```
            "ec2:CreateSnapshot",
            "ec2:CreateSnapshots",
            "ec2:CreateTags",
            "ec2:CreateVolume",
            "ec2:DeleteSnapshot",
            "ec2:DeleteTags",
            "ec2:DeleteVolume",
            "ec2:DescribeInstances",
            "ec2:DescribeSnapshots",
            "ec2:DescribeTags",
            "ec2:DescribeVolumes",
            "ec2:DetachVolume",
            "ebs:ListSnapshotBlocks",
            "ebs:ListChangedBlocks"
        ],
        "Resource": "*"
    }
```

At the same time, edit the role's `Trust relationships` to grant tidb-controller-manager the access to this IAM role.

3. Associate the IAM role with the `ServiceAccount` resources.

```
kubectl annotate sa tidb-backup-manager eks.amazonaws.com/role-arn=arn:
    ↪ aws:iam::123456789012:role/user --namespace=test1
```

When backing up or restoring a TiDB cluster using EBS volume snapshots, you need to associate the IAM role with the `ServiceAccount` resources of tidb-controller-manager.

```
shell kubectl annotate sa tidb-controller-manager eks.amazonaws.com/
↪ role-arn=arn:aws:iam::123456789012:role/user --namespace=tidb-admin
```

Restart the tidb-controller-manager Pod of TiDB Operator to make the configured `ServiceAccount` take effect.

4. Associate the `ServiceAccount` with the TiKV Pod:

```
kubectl patch tc demo1 -n test1 --type merge -p '{"spec":{"tikv":{"
    ↪ serviceAccount": "tidb-backup-manager"}}}'
```

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.

5. (Optional) If your cluster includes TiFlash Pods, repeat step 4 to associate the `ServiceAccount` with the TiFlash Pod.

```
kubectl patch tc demo1 -n test1 --type merge -p '{"spec":{"tiflash":{"
    ↪ serviceAccount": "tidb-backup-manager"}}}'
```

Note:

`arn:aws:iam::123456789012:role/user` is the IAM role created in Step 2.

7.4.3.2 GCS account permissions

7.4.3.2.1 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 Google Cloud console. Refer to Google Cloud documentation for details.

```
kubectl create secret generic gcs-secret --from-file=credentials=./google-
    ↪ credentials.json -n test1
```

7.4.3.3 Azure account permissions

Azure provides different methods to grant permissions for different types of Kubernetes clusters. This document describes the following two methods.

7.4.3.3.1 Grant permissions by access key

The Azure client can read `AZURE_STORAGE_ACCOUNT` and `AZURE_STORAGE_KEY` from the process environment variables to obtain the associated user or role permissions.

Run the following command to create the `azblob-secret` secret and use your Azure account access key to grant permissions. The secret stores the credential used for accessing Azure Blob Storage.

```
kubectl create secret generic azblob-secret --from-literal=
    ↪ AZURE_STORAGE_ACCOUNT=xxx --from-literal=AZURE_STORAGE_KEY=yyy --
    ↪ namespace=test1
```

7.4.3.3.2 Grant permissions by Azure AD

The Azure client can read `AZURE_STORAGE_ACCOUNT`, `AZURE_CLIENT_ID`, `AZURE_TENANT_ID` ↪ , and `AZURE_CLIENT_SECRET` to obtain the associated user or role permissions.

1. Create the `azblob-secret-ad` secret by running the following command. Use the Active Directory (AD) of your Azure account. The secret stores the credential used for accessing Azure Blob Storage.

```
kubectl create secret generic azblob-secret-ad --from-literal=
    ↪ AZURE_STORAGE_ACCOUNT=xxx --from-literal=AZURE_CLIENT_ID=yyy --
    ↪ from-literal=AZURE_TENANT_ID=zzz --from-literal=
    ↪ AZURE_CLIENT_SECRET=aaa --namespace=test1
```

2. Associate the secret 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 Azure Blob Storage as the BR Pod does. Therefore, you need to associate the TiKV Pod with the secret.

```
kubectl patch tc demo1 -n test1 --type merge -p '{"spec":{"tikv":{"
    ↪ envFrom":[{"secretRef":{"name":"azblob-secret-ad"}}]}}}'
```

   After the TiKV Pod is restarted, check whether the Pod has the environment variables.

7.4.4 Amazon S3 Compatible Storage

7.4.4.1 Back up Data to S3-Compatible Storage Using BR

This document describes how to back up the data of a TiDB cluster on AWS Kubernetes to AWS storage. There are two backup types:

- **Snapshot backup**. With snapshot backup, you can restore a TiDB cluster to the time point of the snapshot backup using full restoration.
- **Log backup**. With snapshot backup and log backup, you can restore a TiDB cluster to any point in time. This is also known as Point-in-Time Recovery (PITR).

The backup method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used to get the backup data of the TiDB cluster, and then send the data to the AWS storage. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

7.4.4.1.1 Usage scenarios

If you have the following backup needs, you can use BR's **snapshot backup** method to make an ad-hoc backup or scheduled snapshot backup of the TiDB cluster data to S3-compatible storages.

- To back up a large volume of data (more than 1 TB) at a fast speed
- To get a direct backup of data as SST files (key-value pairs)

If you have the following backup needs, you can use BR **log backup** to make an ad-hoc backup of the TiDB cluster data to S3-compatible storages (you can combine log backup and snapshot backup to restore data more efficiently):

- To restore data of any point in time to a new cluster
- The recovery point object (RPO) is within several minutes.

For other backup needs, refer to Backup and Restore Overview to choose an appropriate backup method.

Note:

- Snapshot backup is only applicable to TiDB v3.1 or later releases.
- Log backup is only applicable to TiDB v6.3 or later releases.
- Data that is backed up using BR can only be restored to TiDB instead of other databases.

7.4.4.1.2 Ad-hoc backup

Ad-hoc backup includes snapshot backup and log backup. For log backup, you can start or stop a log backup task and clean log backup data.

To get an Ad-hoc backup, you need to create a `Backup` Custom Resource (CR) object to describe the backup details. Then, 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 an example about how to back up the data of the `demo1` TiDB cluster in the `test1` Kubernetes namespace to the AWS storage. The following are the detailed steps.

Prerequisites: Prepare for an ad-hoc backup

1. Create a namespace for managing backup. The following example creates a `backup-test` namespace:

```
kubectl create namespace backup-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `backup-test` namespace:

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

3. Grant permissions to the remote storage for the created `backup-test` namespace.

    - If you are using Amazon S3 to backup your cluster, you can grant permissions in three methods. For more information, refer to AWS account permissions.

- If you are using other S3-compatible storage (such as Ceph and MinIO) to backup your cluster, you can grant permissions by using AccessKey and SecretKey.

4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.

    1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the backup database so that the `Backup` CR can adjust the GC time before and after the backup.
    2. Create the `backup-demo1-tidb-secret` secret to store the account and password to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-
    ↪ literal=password=${password} --namespace=test1
```

Snapshot backup

Depending on which method you choose to grant permissions to the remote storage when preparing for the ad-hoc backup, export your data to the S3-compatible storage by doing one of the following:

- Method 1: If you grant permissions by importing AccessKey and SecretKey, create the `Backup` CR to back up cluster data as described below:

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

The content of `full-backup-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: backup-test
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
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

- Method 2: If you grant permissions by associating IAM with Pod, create the `Backup` CR to back up cluster data as described below:

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

The content of `full-backup-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: backup-test
  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
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

- Method 3: If you grant permissions by associating IAM with ServiceAccount, create the `Backup` CR to back up cluster data as described below:

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

The content of `full-backup-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: backup-test
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
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

When configuring `full-backup-s3.yaml`, note the following:

- 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.
- You can ignore the `acl`, `endpoint`, `storageClass` configuration items of Amazon S3. For more information about S3-compatible storage configuration, refer to S3 storage fields.
- Some parameters in `.spec.br` are optional, such as `logLevel` and `statusAddr`. For more information about BR configuration, refer to BR fields.
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.tikvGCLifeTime` and `spec.from` fields in the `Backup` CR.

- For more information about the `Backup` CR fields, refer to Backup CR fields.

View the snapshot backup status

After you create the `Backup` CR, TiDB Operator starts the backup automatically. You can view the backup status by running the following command:

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

From the output, you can find the following information for the `Backup` CR named `demo1-full-backup-s3`. The `COMMITTS` field indicates the time point of the snapshot backup:

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

Log backup

You can use a `Backup` CR to describe the start and stop of a log backup task and manage the log backup data. Log backup grants permissions to remote storages in the same way as snapshot backup. In this section, the example shows log backup operations by taking a `Backup` CR named `demo1-log-backup-s3` as an example. Note that these operations assume that permissions to remote storages are granted using accessKey and secretKey. See the following detailed steps.

Description of the `logSubcommand` field

In the Backup Custom Resource (CR), you can use the `logSubcommand` field to control the state of a log backup task. The `logSubcommand` field supports the following commands:

- `log-start`: initiates a new log backup task or resumes a paused task. Use this command to start the log backup process or resume a task from a paused state.
- `log-pause`: temporarily pauses the currently running log backup task. After pausing, you can use the `log-start` command to resume the task.
- `log-stop`: permanently stops the log backup task. After executing this command, the Backup CR enters a stopped state and cannot be restarted.

These commands provide fine-grained control over the lifecycle of log backup tasks, enabling you to start, pause, resume, and stop tasks effectively to manage log data retention in Kubernetes environments.

In TiDB Operator v1.5.4, v1.6.0, and earlier versions, you can use the `logStop: true/false` field to stop or start log backup tasks. This field is retained for backward compatibility.

However, do not use `logStop` and `logSubcommand` fields in the same Backup CR, as this is not supported. For TiDB Operator v1.5.5, v1.6.1, and later versions, it is recommended to use the `logSubcommand` field to ensure clear and consistent configuration.

Start log backup

1. In the `backup-test` namespace, create a `Backup` CR named `demo1-log-backup-s3`.

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

The content of `log-backup-s3.yaml` is as follows:

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

2. Wait for the start operation to complete:

```
kubectl get jobs -n backup-test
```

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

3. View the newly created `Backup` CR:

```
kubectl get backup -n backup-test
```

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

View the log backup status

You can view the log backup status by checking the information of the `Backup` CR:

```
kubectl describe backup -n backup-test
```

From the output, you can find the following information for the `Backup` CR named `demo1 ↪ -log-backup-s3`. The `Log Checkpoint Ts` field indicates the latest point in time that can be recovered:

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

Pause log backup

You can pause a log backup task by setting the `logSubcommand` field of the Backup Custom Resource (CR) to `log-pause`. The following example shows how to pause the `demo1-log-backup-s3` CR created in Start log backup.

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

To pause the log backup task, change the value of `logSubcommand` from `log-start` to `log-pause`, then save and exit the editor. The modified content is as follows:

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

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

You can verify that the `STATUS` of the `demo1-log-backup-s3` Backup CR changes from `Running` to `Pause`:

```
kubectl get backup -n backup-test
```

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

Resume log backup

If a log backup task is paused, you can resume it by setting the `logSubcommand` field to `log-start`. The following example shows how to resume the `demo1-log-backup-s3` CR that was paused in Pause Log Backup.

> **Note:**
>
> This operation applies only to tasks in the `Pause` state. You cannot resume tasks in the `Fail` or `Stopped` state.

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

To resume the log backup task, change the value of `logSubcommand` from `log-pause` to `log-start`, then save and exit the editor. The modified content is as follows:

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

```
    clusterNamespace: test1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
```

You can verify that the `STATUS` of the `demo1-log-backup-s3` Backup CR changes from `Pause` to `Running`:

```
kubectl get backup -n backup-test
```

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

Stop log backup

You can stop a log backup task by setting the `logSubcommand` field of the Backup Custom Resource (CR) to `log-stop`. The following example shows how to stop the `demo1-log-backup-s3` CR created in Start log backup.

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

Change the value of `logSubcommand` to `log-stop`, then save and exit the editor. The modified content is as follows:

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

You can verify that the `STATUS` of the `Backup` CR named `demo1-log-backup-s3` changes from `Running` to `Stopped`:

```
kubectl get backup -n backup-test
```

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

`Stopped` is the terminal state for log backup. In this state, you cannot change the backup state again, but you can still clean up the log backup data.

In TiDB Operator v1.5.4, v1.6.0, and earlier versions, you can use the `logStop: true /false` field to stop or start log backup tasks. This field is retained for backward compatibility.

Clean log backup data

1. Because you already created a `Backup` CR named `demo1-log-backup-s3` when you started log backup, you can clean the log data backup by modifying the same `Backup` CR. The following example shows how to clean log backup data generated before 2022-10-10T15:21:00+08:00.

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

   In the last line of the CR, append `spec.logTruncateUntil: "2022-10-10T15:21:00+08:00"`. Then save and quit the editor. The modified content is as follows:

   ```
   ---
   apiVersion: pingcap.com/v1alpha1
   kind: Backup
   metadata:
     name: demo1-backup-s3
     namespace: backup-test
   spec:
     backupMode: log
     logSubcommand: log-start/log-pause/log-stop
     br:
       cluster: demo1
       clusterNamespace: test1
       sendCredToTikv: true
     s3:
       provider: aws
       secretName: s3-secret
       region: us-west-1
       bucket: my-bucket
       prefix: my-log-backup-folder
     logTruncateUntil: "2022-10-10T15:21:00+08:00"
   ```

2. Wait for the clean operation to complete:

```
kubectl get jobs -n backup-test
```

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

3. View the Backup CR information:

```
kubectl describe backup -n backup-test
```

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

You can also view the information by running the following command:

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

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

Backup CR examples

Back up data of all clusters

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

7.4.4.1.3 Scheduled snapshot 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 snapshot backup is described by a custom `BackupSchedule` CR object. A snapshot backup is triggered at each backup time point. Its underlying implementation is the ad-hoc snapshot backup.

Prerequisites: Prepare for a scheduled snapshot backup

The steps to prepare for a scheduled snapshot backup are the same as that of Prepare for an ad-hoc backup.

Perform a scheduled snapshot backup

Depending on which method you choose to grant permissions to the remote storage, perform a scheduled snapshot backup by doing one of the following:

- Method 1: 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: backup-test
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
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

- Method 2: 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: backup-test
  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
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

- Method 3: 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: backup-test
```

```
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
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder
```

In the above example of `backup-scheduler-aws-s3.yaml`, the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`.

- For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.
- `backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR.

After creating the scheduled snapshot 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
```

7.4.4.1.4 Integrated management of scheduled snapshot backup and log backup

You can use the `BackupSchedule` CR to integrate the management of scheduled snapshot backup and log backup for TiDB clusters. By setting the backup retention time, you can regularly recycle the scheduled snapshot backup and log backup, and ensure that you can perform PITR recovery through the scheduled snapshot backup and log backup within the retention period.

The following example creates a `BackupSchedule` CR named `integrated-backup-schedule-s3`. In the example, accessKey and secretKey are used to access the remote storage. For more information about the authorization method, refer to AWS account permissions.

Prerequisites: Prepare for a scheduled snapshot backup environment

The steps to prepare for a scheduled snapshot backup are the same as those of Prepare for an ad-hoc backup.

Create `BackupSchedule`

1. Create a `BackupSchedule` CR named `integrated-backup-schedule-s3` in the `backup-test` namespace.

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

The content of `integrated-backup-schedule-s3.yaml` is as follows:

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

```
    prefix: my-folder-snapshot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      clusterNamespace: test1
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder-log
```

In the above example of `integrated-backup-schedule-s3.yaml`, the `backupSchedule` ↪ configuration consists of three parts: the unique configuration of `backupSchedule`, the configuration of the snapshot backup `backupTemplate`, and the configuration of the log backup `logBackupTemplate`.

For the field description of `backupSchedule`, refer to BackupSchedule CR fields.

2. After creating `backupSchedule`, use the following command to check the backup status:

```
kubectl get bks -n backup-test -o wide
```

A log backup task is created together with `backupSchedule`. You can check the log backup name through the `status.logBackup` field of the `backupSchedule` CR.

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

3. To perform data restoration for a cluster, you need to specify the backup path. You can use the following command to check all the backup items under the scheduled snapshot backup.

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

The `MODE` field in the output indicates the backup mode. `snapshot` indicates the scheduled snapshot backup, and `log` indicates the log backup.

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

7.4.4.1.5 Delete the backup CR

If you no longer need the backup CR, refer to Delete the Backup CR.

7.4.4.1.6 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

7.4.4.2 Restore Data from S3-Compatible Storage Using BR

This document describes how to restore the backup data stored in S3-compatible storages to a TiDB cluster on Kubernetes, including two restoration methods:

- Full restoration. This method takes the backup data of snapshot backup and restores a TiDB cluster to the time point of the snapshot backup.
- Point-in-time recovery (PITR). This method takes the backup data of both snapshot backup and log backup and restores a TiDB cluster to any point in time.

The restore method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used to restore the data. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

PITR allows you to restore a new TiDB cluster to any point in time of the backup cluster. To use PITR, you need the backup data of snapshot backup and log backup. During the restoration, the snapshot backup data is first restored to the TiDB cluster, and then the log backup data between the snapshot backup time point and the specified point in time is restored to the TiDB cluster.

Note:

- BR is only applicable to TiDB v3.1 or later releases.
- PITR is only applicable to TiDB v6.3 or later releases.
- Data restored by BR cannot be replicated to a downstream cluster, because BR directly imports SST and LOG files to TiDB and the downstream cluster currently cannot access the upstream SST and LOG files.

7.4.4.2.1 Full restoration

This document provides an example about how to restore the backup data from the `spec.s3.prefix` folder of the `spec.s3.bucket` bucket on Amazon S3 to the `demo2` TiDB cluster in the `test2` namespace. The following are the detailed steps.

Prerequisites: Complete the snapshot backup

In this example, the `my-full-backup-folder` folder in the `my-bucket` bucket of Amazon S3 stores the snapshot backup data. For steps of performing snapshot backup, refer to Back up Data to S3 Using BR.

Step 1: Prepare the restore environment

Before restoring backup data on a S3-compatible storage to TiDB using BR, take the following steps to prepare the restore environment:

1. Create a namespace for managing restoration. The following example creates a `restore-test` namespace:

```
kubectl create namespace restore-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `restore-test` namespace:

```
kubectl apply -f backup-rbac.yaml -n restore-test
```

3. Grant permissions to the remote storage for the `restore-test` namespace.

   If the data to be restored is in Amazon S3, you can grant permissions in three methods. For more information, see AWS account permissions.

   If the data to be restored is in other S3-compatible storage (such as Ceph and MinIO), you can grant permissions by using AccessKey and SecretKey.

4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.

   1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the target database so that the `Restore` CR can adjust the GC time before and after the restore.
   2. Create the `restore-demo2-tidb-secret` secret to store the account and password to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-
    ↪ literal=password=${password} --namespace=test2
```

Step 2: Restore the backup data to a TiDB cluster

Depending on which method you choose to grant permissions to the remote storage when preparing the restore environment, you can restore the data by doing one of the following:

- Method 1: If you grant permissions by importing AccessKey and SecretKey, create the `Restore` CR to restore cluster data as described below:

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

The content of `restore-full-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: restore-test
spec:
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

- Method 2: If you grant permissions by associating IAM with Pod, create the `Restore` CR to restore cluster data as described below:

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

The content of `restore-full-s3.yaml` is as follows:

“‘yaml

```
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: restore-test
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  br:
    cluster: demo2
    sendCredToTikv: false
    clusterNamespace: test2
```

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

- Method 3: If you grant permissions by associating IAM with ServiceAccount, create the `Restore` CR to restore cluster data as described below:

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

The content of `restore-full-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: restore-test
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
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-full-backup-folder
```

When configuring `restore-full-s3.yaml`, note the following:

- For more information about S3-compatible storage configuration, refer to S3 storage fields.
- Some parameters in `.spec.br` are optional, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.to` fields in the `Restore` CR.
- For more information about the `Restore` CR fields, refer to Restore CR fields.

After creating the `Restore` CR, execute the following command to check the restore status:

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

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

7.4.4.2.2 Point-in-time recovery

This section provides an example about how to perform point-in-time recovery (PITR) in a `demo3` cluster in the `test3` namespace. PITR takes two steps:

1. Restore the cluster to the time point of the snapshot backup using the snapshot backup data in the `spec.pitrFullBackupStorageProvider.s3.prefix` folder of the `spec.pitrFullBackupStorageProvider.s3.bucket` bucket.
2. Restore the cluster to any point in time using the log backup data in the `spec.s3.prefix` folder of the `spec.s3.bucket` bucket.

The detailed steps are as follows.

Prerequisites: Complete data backup

In this example, the `my-bucket` bucket of Amazon S3 stores the following two types of backup data:

- The snapshot backup data generated during the **log backup**, stored in the `my-full-backup-folder-pitr` folder.
- The log backup data, stored in the `my-log-backup-folder-pitr` folder.

For detailed steps of how to perform data backup, refer to Back up data to Azure Blob Storage.

Note:

The specified restoration time point must be between the snapshot backup time point and the log backup `checkpoint-ts`.

Step 1: Prepare the restoration environment

Before restoring backup data on S3-compatible storages to TiDB using BR, take the following steps to prepare the restoration environment:

1. Create a namespace for managing restoration. The following example creates a `restore-test` namespace:

```
kubectl create namespace restore-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `restore-test` namespace:

```
kubectl apply -f backup-rbac.yaml -n restore-test
```

3. Grant permissions to the remote storage for the `restore-test` namespace.

   If the data to be restored is in Amazon S3, you can grant permissions in three methods. For more information, see AWS account permissions.

   If the data to be restored is in other S3-compatible storage (such as Ceph and MinIO), you can grant permissions by using AccessKey and SecretKey.

Step 2: Restore the backup data to a TiDB cluster

The example in this section restores the snapshot backup data to the cluster. The specified restoration time point must be between the time point of snapshot backup and the `Log Checkpoint Ts` of log backup.

PITR grants permissions to remote storages in the same way as snapshot backup. The example in this section grants permissions by using AccessKey and SecretKey.

The detailed steps are as follows:

1. Create a `Restore` CR named `demo3-restore-s3` in the `restore-test` namespace and specify the restoration time point as `2022-10-10T17:21:00+08:00`:

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

   The content of `restore-point-s3.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-s3
  namespace: restore-test
spec:
  restoreMode: pitr
  br:
    cluster: demo3
    clusterNamespace: test3
  s3:
    provider: aws
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder-pitr
  pitrRestoredTs: "2022-10-10T17:21:00+08:00"
  pitrFullBackupStorageProvider:
    s3:
      provider: aws
      region: us-west-1
      bucket: my-bucket
      prefix: my-full-backup-folder-pitr
```

When you configure `restore-point-s3.yaml`, note the following:

- `spec.restoreMode`: when you perform PITR, set this field to `pitr`. The default value of this field is `snapshot`, which means snapshot backup.

2. Wait for the restoration operation to complete:

```
kubectl get jobs -n restore-test
```

```
NAME                      COMPLETIONS  ...
restore-demo3-restore-s3 1/1          ...
```

You can also check the restoration status by using the following command:

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

```
NAME             STATUS    ...
demo3-restore-s3 Complete  ...
```

7.4.4.2.3 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

7.4.4.3 Back up Data to S3-Compatible Storage Using Dumpling

This document describes how to back up the data of the TiDB cluster on Kubernetes to an S3-compatible storage. "Backup" in this document refers to full backup (ad-hoc full backup and scheduled full backup).

The backup method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator v1.1 or later versions. 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.

Dumpling is a data export tool that exports data stored in TiDB/MySQL as SQL or CSV files and can be used to make a logical full backup or export.

7.4.4.3.1 Usage scenarios

You can use the backup method described in this document if you want to make an ad-hoc full backup or scheduled full backup of the TiDB cluster data to S3-compatible storages with the following needs:

- To export SQL or CSV files
- To limit the memory usage of a single SQL statement
- To export the historical data snapshot of TiDB

7.4.4.3.2 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

Before you use Dumpling to back up the TiDB cluster data to the S3-compatible storage, make sure that you have the following 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'@'%';
```

Step 1: Prepare 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/v1.6.1/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
   ```

Step 2: Perform ad-hoc backup

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
```

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
- Method 1: 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
```

- Method 2: 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
```

- Method 3: 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
```

- Method 4: 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.

7.4.4.3.3 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.

Step 1: Prepare for scheduled backup

The prerequisites for the scheduled backup is the same as the prepare for ad-hoc full backup.

Step 2: Perform scheduled backup

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
```

- Method 1: 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
```

- Method 2: 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
```

- Method 3: 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
```

- Method 4: 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.

7.4.4.3.4 Delete the backup CR

After the backup, you might need to delete the backup CR. For details, refer to Delete the Backup CR.

7.4.4.3.5 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

7.4.4.4 Restore Data from S3-Compatible Storage Using TiDB Lightning

This document describes how to restore the TiDB cluster data backed up using TiDB Operator on 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 is a tool used for fast full import of large amounts of data into a TiDB cluster. It reads data from local disks, Google Cloud Storage (GCS) or Amazon S3. TiDB Lightning supports three backends: `Importer-backend`, `Local-backend`, and `TiDB-backend` ↪ . In this document, `TiDB-backend` is used. 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 the S3-compatible storage is restored to the TiDB cluster.

7.4.4.4.1 Usage scenarios

You can use the restore solution introduced in this document if you need to export the backup data from S3 to a TiDB cluster, with the following requirements:

- To restore data with lower resource usage and lower network bandwidth usage. A restore speed of 50 GB/h is acceptable.
- To import data into the cluster with ACID compliance.
- The TiDB cluster can still provide services during the restore process.

7.4.4.4.2 Prerequisites

Before you perform the data restore, you need to prepare the restore environment and get the required database account privileges.

Prepare the restore environment

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
```

Get the required database account privileges

Before you use TiDB Lightning to restore the backup data in S3 to the TiDB cluster, make sure that you have the following database account privileges:

Privileges	Scope
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables

7.4.4.4.3 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
```

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

1. Create Restore customer resource (CR) and restore the specified backup data to the TiDB cluster.

    - Method 1: 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
```

- Method 2: 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
```

- Method 3: 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
```

- Method 4: 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
```

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.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.

7.4.4.4.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

7.4.5 Google Cloud Storage

7.4.5.1 Back up Data to GCS Using BR

This document describes how to back up the data of a TiDB cluster on Kubernetes to Google Cloud Storage (GCS). There are two backup types:

- **Snapshot backup**. With snapshot backup, you can restore a TiDB cluster to the time point of the snapshot backup using full restoration.
- **Log backup**. With snapshot backup and log backup, you can restore a TiDB cluster to any point in time. This is also known as Point-in-Time Recovery (PITR).

The backup method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used to get the backup data of the TiDB cluster, and then send the data to the AWS storage. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

7.4.5.1.1 Usage scenarios

If you have the following backup needs, you can use BR's **snapshot backup** method to make an ad-hoc backup or scheduled snapshot backup of the TiDB cluster data to GCS.

- To back up a large volume of data (more than 1 TB) at a fast speed
- To get a direct backup of data as SST files (key-value pairs)

If you have the following backup needs, you can use BR **log backup** to make an ad-hoc backup of the TiDB cluster data to GCS (you can combine log backup and snapshot backup to restore data more efficiently):

- To restore data of any point in time to a new cluster
- The recovery point object (RPO) is within several minutes.

For other backup needs, refer to Backup and Restore Overview to choose an appropriate backup method.

> **Note:**
>
> - Snapshot backup is only applicable to TiDB v3.1 or later releases.
> - Log backup is only applicable to TiDB v6.3 or later releases.
> - Data that is backed up using BR can only be restored to TiDB instead of other databases.

7.4.5.1.2 Ad-hoc backup

Ad-hoc backup includes snapshot backup and log backup. For log backup, you can start or stop a log backup task and clean log backup data.

To get an Ad-hoc backup, you need to create a `Backup` Custom Resource (CR) object to describe the backup details. Then, 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 an example about how to back up the data of the `demo1` TiDB cluster in the `test1` Kubernetes namespace to GCS. The following are the detailed steps.

Prerequisites: Prepare for an ad-hoc backup

1. Create a namespace for managing backup. The following example creates a `backup-test` namespace:

```
kubectl create namespace backup-test
```

2. 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 backup-test
```

3. Grant permissions to the remote storage for the created `backup-test` namespace.

   Refer to GCS account permissions.

4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.

   1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the backup database so that the `Backup` CR can adjust the GC time before and after the backup.
   2. Create the `backup-demo1-tidb-secret` secret to store the root account and password to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-
    ↪ literal=password=<password> --namespace=test1
```

Perform an ad-hoc backup

1. Create the `Backup` CR to back up cluster data to GCS as described below:

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

   The content of `full-backup-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-gcs
  namespace: backup-test
spec:
  # backupType: full
  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: my-bucket
    prefix: my-full-backup-folder
    # location: us-east1
    # storageClass: STANDARD_IA
    # objectAcl: private
```

When configuring the `full-backup-gcs.yaml`, note the following:

- 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.
- Some parameters in `spec.br` are optional, such as `logLevel` and `statusAddr`. For more information about BR configuration, refer to BR fields.
- Some parameters in `spec.gcs` are optional, such as `location`, `objectAcl`, and `storageClass`. For more information about GCS configuration, refer to GCS fields.
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.tikvGCLifeTime` and `spec.from` fields in the `Backup` CR.
- For more information about the `Backup` CR fields, refer to Backup CR fields.

2. After you create the `Backup` CR, TiDB Operator starts the backup automatically. You can view the backup status by running the following command:

```
kubectl get bk -n test1 -owide
```

View the snapshot backup status

After you create the `Backup` CR, TiDB Operator starts the backup automatically. You can view the backup status by running the following command:

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

From the output, you can find the following information for the `Backup` CR named `demo1` ↪ `-full-backup-gcs`. The `COMMITTS` field indicates the time point of the snapshot backup:

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

Log backup

You can use a `Backup` CR to describe the start and stop of a log backup task and manage the log backup data. Log backup grants permissions to remote storages in the same way as snapshot backup. In this section, the example shows how to create a `Backup` CR named `demo1-log-backup-gcs`. See the following detailed steps.

Description of the `logSubcommand` field

In the Backup Custom Resource (CR), you can use the `logSubcommand` field to control the state of a log backup task. The `logSubcommand` field supports the following commands:

- `log-start`: initiates a new log backup task or resumes a paused task. Use this command to start the log backup process or resume a task from a paused state.
- `log-pause`: temporarily pauses the currently running log backup task. After pausing, you can use the `log-start` command to resume the task.
- `log-stop`: permanently stops the log backup task. After executing this command, the Backup CR enters a stopped state and cannot be restarted.

These commands provide fine-grained control over the lifecycle of log backup tasks, enabling you to start, pause, resume, and stop tasks effectively to manage log data retention in Kubernetes environments.

In TiDB Operator v1.5.4, v1.6.0, and earlier versions, you can use the `logStop: true` ↪ `/false` field to stop or start log backup tasks. This field is retained for backward compatibility.

However, do not use `logStop` and `logSubcommand` fields in the same Backup CR, as this is not supported. For TiDB Operator v1.5.5, v1.6.1, and later versions, it is recommended to use the `logSubcommand` field to ensure clear and consistent configuration.

Start log backup

1. In the `backup-test` namespace, create a `Backup` CR named `demo1-log-backup-gcs`.

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

The content of `log-backup-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: backup-test
spec:
  backupMode: log
```

```
  logSubcommand: log-start
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

2. Wait for the start operation to complete:

```
kubectl get jobs -n backup-test
```

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

3. View the newly created `Backup` CR:

```
kubectl get backup -n backup-test
```

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

View the log backup status

You can view the log backup status by checking the information of the `Backup` CR:

```
kubectl describe backup -n backup-test
```

From the output, you can find the following information for the `Backup` CR named `demo1 -log-backup-gcs`. The `Log Checkpoint Ts` field indicates the latest point in time that can be recovered:

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

```
    Status:              True
    Type:                Running
Log Checkpoint Ts:    436569119308644661
```

Pause log backup

You can pause a log backup task by setting the `logSubcommand` field of the Backup Custom Resource (CR) to `log-pause`. The following example shows how to pause the `demo1-log-backup-gcs` CR created in Start log backup.

```
kubectl edit backup demo1-log-backup-gcs -n backup-test
```

To pause the log backup task, change the value of `logSubcommand` from `log-start` to `log-pause`, then save and exit the editor. The modified content is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: backup-test
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

You can verify that the `STATUS` of the `demo1-log-backup-gcs` Backup CR changes from `Running` to `Pause`:

```
kubectl get backup -n backup-test
```

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

Resume log backup

If a log backup task is paused, you can resume it by setting the `logSubcommand` field to `log-start`. The following example shows how to resume the `demo1-log-backup-gcs` CR that was paused in Pause Log Backup.

Note:

This operation applies only to tasks in the `Pause` state. You cannot resume tasks in the `Fail` or `Stopped` state.

```
kubectl edit backup demo1-log-backup-gcs -n backup-test
```

To resume the log backup task, change the value of `logSubcommand` from `log-pause` to `log-start`, then save and exit the editor. The modified content is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: backup-test
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

You can verify that the `STATUS` of the `demo1-log-backup-gcs` Backup CR changes from `Pause` to `Running`:

```
kubectl get backup -n backup-test
```

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

Stop log backup

You can stop a log backup task by setting the `logSubcommand` field of the Backup Custom Resource (CR) to `log-stop`. The following example shows how to stop the `demo1-log-backup-gcs` CR created in Start log backup.

```
kubectl edit backup demo1-log-backup-gcs -n backup-test
```

Change the value of `logSubcommand` to `log-stop`, then save and exit the editor. The modified content is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: backup-test
spec:
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

You can verify that the `STATUS` of the `Backup` CR named `demo1-log-backup-gcs` changes from `Running` to `Stopped`:

```
kubectl get backup -n backup-test
```

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

`Stopped` is the terminal state for log backup. In this state, you cannot change the backup state again, but you can still clean up the log backup data.

In TiDB Operator v1.5.4, v1.6.0, and earlier versions, you can use the `logStop: true /false` field to stop or start log backup tasks. This field is retained for backward compatibility.

Clean log backup data

1. Because you already created a `Backup` CR named `demo1-log-backup-gcs` when you started log backup, you can clean the log data backup by modifying the same `Backup` CR. The following example shows how to clean log backup data generated before 2022-10-10T15:21:00+08:00.

   ```
   kubectl edit backup demo1-log-backup-gcs -n backup-test
   ```

   In the last line of the CR, append `spec.logTruncateUntil: "2022-10-10T15 :21:00+08:00"`. Then save and quit the editor. The modified content is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: backup-test
spec:
  backupMode: log
  logSubcommand: log-start/log-pause/log-stop
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
  logTruncateUntil: "2022-10-10T15:21:00+08:00"
```

2. Wait for the clean operation to complete:

```
kubectl get jobs -n backup-test
```

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

3. View the `Backup` CR information:

```
kubectl describe backup -n backup-test
```

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

You can also view the information by running the following command:

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

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

Backup CR examples

Back up data of all clusters

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: backup-test
spec:
  # backupType: full
  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: backup-test
spec:
  # backupType: full
  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: backup-test
spec:
  # backupType: full
  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: backup-test
spec:
  # backupType: full
  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
```

7.4.5.1.3 Scheduled snapshot 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 snapshot backup is described by a custom `BackupSchedule` CR object. A snapshot backup is triggered at each backup time point. Its underlying implementation is the ad-hoc snapshot backup.

Prerequisites: Prepare for a scheduled snapshot backup

The steps to prepare for a scheduled snapshot backup are the same as that of Prepare for an ad-hoc backup.

Perform a scheduled snapshot backup

1. Create a `BackupSchedule` CR to 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: backup-test
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      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
```

From the above in `backup-schedule-gcs.yaml`, you can see that the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule` ↪ , and the other is `backupTemplate`.

- For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.
- `backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR.

2. After creating the scheduled snapshot backup, use the following command to check the backup status:

```
kubectl get bks -n backup-test -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 backup-test
```

7.4.5.1.4 Integrated management of scheduled snapshot backup and log backup

You can use the `BackupSchedule` CR to integrate the management of scheduled snapshot backup and log backup for TiDB clusters. By setting the backup retention time, you can regularly recycle the scheduled snapshot backup and log backup, and ensure that you can perform PITR recovery through the scheduled snapshot backup and log backup within the retention period.

The following example creates a `BackupSchedule` CR named `integrated-backup-` ↪ `schedule-gcs`. For more information about the authorization method, refer to GCS account permissions.

Prerequisites: Prepare for a scheduled snapshot backup environment

The steps to prepare for a scheduled snapshot backup are the same as those of Prepare for an ad-hoc backup.

Create `BackupSchedule`

1. Create a `BackupSchedule` CR named `integrated-backup-schedule-gcs` in the `backup-test` namespace.

```
kubectl apply -f integrated-backup-scheduler-gcs.yaml
```

The content of `integrated-backup-schedule-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-gcs
  namespace: backup-test
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      clusterNamespace: test1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-snapshot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      clusterNamespace: test1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-log
```

In the above example of `integrated-backup-scheduler-gcs.yaml`, the `backupSchedule` ↪ configuration consists of three parts: the unique configuration of `backupSchedule`, the configuration of the snapshot backup `backupTemplate`, and the configuration of the log backup `logBackupTemplate`.

For the field description of `backupSchedule`, refer to BackupSchedule CR fields.

2. After creating `backupSchedule`, use the following command to check the backup status:

```
kubectl get bks -n backup-test -o wide
```

A log backup task is created together with `backupSchedule`. You can check the log backup name through the `status.logBackup` field of the `backupSchedule` CR.

```
kubectl describe bks integrated-backup-schedule-gcs -n backup-test
```

3. To perform data restoration for a cluster, you need to specify the backup path. You can use the following command to check all the backup items under the scheduled snapshot backup.

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

The `MODE` field in the output indicates the backup mode. `snapshot` indicates the scheduled snapshot backup, and `log` indicates the log backup.

```
NAME                                                MODE      STATUS
    ↪ ....
integrated-backup-schedule-gcs-2023-03-08t02-50-00  snapshot  Complete
    ↪ ....
log-integrated-backup-schedule-gcs                  log       Running
    ↪ ....
```

7.4.5.1.5 Delete the backup CR

If you no longer need the backup CR, refer to Delete the Backup CR.

7.4.5.1.6 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

7.4.5.2 Restore Data from GCS Using BR

This document describes how to restore the backup data stored in Google Cloud Storage (GCS) to a TiDB cluster on Kubernetes, including two restoration methods:

- Full restoration. This method takes the backup data of snapshot backup and restores a TiDB cluster to the time point of the snapshot backup.
- Point-in-time recovery (PITR). This method takes the backup data of both snapshot backup and log backup and restores a TiDB cluster to any point in time.

The restore method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used to restore the data. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

PITR allows you to restore a new TiDB cluster to any point in time of the backup cluster. To use PITR, you need the backup data of snapshot backup and log backup. During the restoration, the snapshot backup data is first restored to the TiDB cluster, and then the log backup data between the snapshot backup time point and the specified point in time is restored to the TiDB cluster.

Note:

- BR is only applicable to TiDB v3.1 or later releases.
- PITR is only applicable to TiDB v6.3 or later releases.
- Data restored by BR cannot be replicated to a downstream cluster, because BR directly imports SST and LOG files to TiDB and the downstream cluster currently cannot access the upstream SST and LOG files.

This document provides an example about how to restore the backup data from the `spec.gcs.prefix` folder of the `spec.gcs.bucket` bucket on GCS to the `demo2` TiDB cluster in the `test2` namespace. The following are the detailed steps.

7.4.5.2.1 Full restoration

This section provides an example about how to restore the backup data from the `spec.gcs.prefix` folder of the `spec.gcs.bucket` bucket on GCS to the `demo2` TiDB cluster in the `test2` namespace. The following are the detailed steps.

Prerequisites: Complete the snapshot backup

In this example, the `my-full-backup-folder` folder in the `my-bucket` bucket of GCS stores the snapshot backup data. For steps of performing snapshot backup, refer to Back up Data to GCS Using BR.

Step 1: Prepare the restore environment

Before restoring backup data on GCS to TiDB using BR, take the following steps to prepare the restore environment:

1. Create a namespace for managing restoration. The following example creates a `restore-test` namespace:

```
kubectl create namespace restore-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `restore-test` namespace:

```
kubectl apply -f backup-rbac.yaml -n restore-test
```

3. Grant permissions to the remote storage for the `restore-test` namespace.

   Refer to GCS account permissions.

4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.

   1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the target database so that the `Restore` CR can adjust the GC time before and after the restore.
   2. Create the `restore-demo2-tidb-secret` secret to store the root account and password to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-
    ↪ literal=user=root --from-literal=password=<password> --
    ↪ namespace=test2
```

Step 2: Restore the backup data to a TiDB cluster

1. Create the `Restore` custom resource (CR) to restore the specified data to your cluster:

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

   The content of `restore-full-gcs.yaml` file is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-gcs
  namespace: restore-test
spec:
  # backupType: full
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status-addr}
    # concurrency: 4
    # rateLimit: 0
    # checksum: true
```

```
  # sendCredToTikv: true
gcs:
  projectId: ${project_id}
  secretName: gcs-secret
  bucket: my-bucket
  prefix: my-full-backup-folder
  # location: us-east1
  # storageClass: STANDARD_IA
  # objectAcl: private
```

When configuring `restore-full-gcs.yaml`, note the following:

- For more information about GCS configuration, refer to GCS fields.
- Some parameters in `.spec.br` are optional, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.to` fields in the `Restore` CR.
- 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:

```
kubectl get restore -n restore-test -owide
```

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

7.4.5.2.2 Point-in-time recovery

This section provides an example about how to perform point-in-time recovery (PITR) in a `demo3` cluster in the `test3` namespace. PITR takes two steps:

1. Restore the cluster to the time point of the snapshot backup using the snapshot backup data in the `spec.pitrFullBackupStorageProvider.gcs.prefix` folder of the `spec.pitrFullBackupStorageProvider.gcs.bucket` bucket.
2. Restore the cluster to any point in time using the log backup data in the `spec.gcs.prefix` folder of the `spec.gcs.bucket` bucket.

The detailed steps are as follows.

Prerequisites: Complete data backup

In this example, the `my-bucket` bucket of GCS stores the following two types of backup data:

- The snapshot backup data generated during the **log backup**, stored in the `my-full-backup-folder-pitr` folder.

- The log backup data, stored in the `my-log-backup-folder-pitr` folder.

For detailed steps of how to perform data backup, refer to Back up data to GCS Using BR.

> **Note:**
>
> The specified restoration time point must be between the snapshot backup time point and the log backup `checkpoint-ts`.

Step 1: Prepare the restoration environment

Before restoring backup data on GCS to TiDB using BR, take the following steps to prepare the restoration environment:

1. Create a namespace for managing restoration. The following example creates a `restore-test` namespace:

```
kubectl create namespace restore-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `restore-test` namespace:

```
kubectl apply -f backup-rbac.yaml -n restore-test
```

3. Grant permissions to the remote storage for the `restore-test` namespace.

   Refer to GCS account permissions.

Step 2: Restore the backup data to a TiDB cluster

The example in this section restores the snapshot backup data to the cluster. The specified restoration time point must be between the time point of snapshot backup and the `Log Checkpoint Ts` of log backup.

The detailed steps are as follows:

1. Create a `Restore` CR named `demo3-restore-gcs` in the `restore-test` namespace and specify the restoration time point as `2022-10-10T17:21:00+08:00`:

```
kubectl apply -f restore-point-gcs.yaml
```

   The content of `restore-point-gcs.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-gcs
  namespace: restore-test
spec:
  restoreMode: pitr
  br:
    cluster: demo3
    clusterNamespace: test3
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder-pitr
  pitrRestoredTs: "2022-10-10T17:21:00+08:00"
  pitrFullBackupStorageProvider:
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: my-full-backup-folder-pitr
```

When you configure `restore-point-gcs.yaml`, note the following:

- `spec.restoreMode`: when you perform PITR, set this field to `pitr`. The default value of this field is `snapshot`, which means snapshot backup.

2. Wait for the restoration operation to complete:

```
kubectl get jobs -n restore-test
```

```
NAME                       COMPLETIONS  ...
restore-demo3-restore-gcs 1/1          ...
```

You can also check the restoration status by using the following command:

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

```
NAME               STATUS    ...
demo3-restore-gcs Complete ...
```

7.4.5.2.3 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

7.4.5.3 Back up Data to GCS Using Dumpling

This document describes how to back up the data of the TiDB cluster on Kubernetes to Google Cloud Storage (GCS). "Backup" in this document refers to full backup (ad-hoc full backup and scheduled full backup).

The backup method described in this document is implemented using CustomResourceDefinition (CRD) in TiDB Operator v1.1 or later versions. Dumpling is used to get the logic backup of the TiDB cluster, and then this backup data is sent to the remote GCS.

Dumpling is a data export tool that exports data stored in TiDB/MySQL as SQL or CSV files and can be used to make a logical full backup or export.

7.4.5.3.1 Usage scenarios

You can use the backup method described in this document if you want to make an ad-hoc full backup or scheduled full backup of the TiDB cluster data to GCS with the following needs:

- To export SQL or CSV files
- To limit the memory usage of a single SQL statement
- To export the historical data snapshot of TiDB

7.4.5.3.2 Prerequisites

Before you use Dumpling to back up the TiDB cluster data to GCS, make sure that you have the following 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

7.4.5.3.3 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.

Step 1: Prepare 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
```

Step 2: Perform ad-hoc backup

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
```

7.4.5.3.4 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.

Step 1: Prepare for scheduled backup

The preparation for the scheduled backup is the same as the prepare for ad-hoc full backup.

Step 2: Perform scheduled 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:
    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.

7.4.5.3.5 Delete the backup CR

After the backup, you might need to delete the backup CR. For details, refer to Delete the Backup CR.

7.4.5.3.6 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

7.4.5.4 Restore Data from GCS

This document describes how to restore the TiDB cluster data backed up using TiDB Operator on 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 is a tool used for fast full import of large amounts of data into a TiDB cluster. It reads data from local disks, Google Cloud Storage (GCS) or Amazon S3. TiDB Lightning supports three backends: `Importer-backend`, `Local-backend`, and `TiDB-backend` ↪ . In this document, `TiDB-backend` is used. 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 GCS is restored to the TiDB cluster.

7.4.5.4.1 Usage scenarios

You can use the restore solution introduced in this document if you need to export the backup data from GCS to a TiDB cluster, with the following requirements:

- To restore data with lower resource usage and lower network bandwidth usage. A restore speed of 50 GB/h is acceptable.
- To import data into the cluster with ACID compliance.
- The TiDB cluster can still provide services during the restore process.

7.4.5.4.2 Prerequisites

Before you perform the data restore, you need to prepare the restore environment and get the required database account privileges.

Prepare the restore environment

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
```

Get the required database account privileges

Before you use TiDB Lightning to restore the backup data in GCS to the TiDB cluster, make sure that you have the following database account privileges:

Privileges	Scope
SELECT	Tables
INSERT	Tables
UPDATE	Tables
DELETE	Tables
CREATE	Databases, tables
DROP	Databases, tables
ALTER	Tables

7.4.5.4.3 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.

7.4.5.4.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

7.4.6 Azure Blob Storage

7.4.6.1 Back up Data to Azure Blob Storage Using BR

This document describes how to back up the data of a TiDB cluster on Kubernetes to Azure Blob Storage. There are two backup types:

- **Snapshot backup**. With snapshot backup, you can restore a TiDB cluster to the time point of the snapshot backup using full restoration.
- **Log backup**. With snapshot backup and log backup, you can restore a TiDB cluster to any point in time. This is also known as Point-in-Time Recovery (PITR).

The backup method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used to get the backup data of the TiDB cluster, and then send the data to Azure Blob Storage. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

7.4.6.1.1 Usage scenarios

If you have the following backup needs, you can use BR to make an ad-hoc backup or scheduled snapshot backup of the TiDB cluster data to Azure Blob Storage.

- To back up a large volume of data (more than 1 TB) at a fast speed.
- To get a direct backup of data as SST files (key-value pairs).

If you have the following backup needs, you can use BR **log backup** to make an ad-hoc backup of the TiDB cluster data to Azure Blob Storage (you can combine log backup and snapshot backup to restore data more efficiently):

- To restore data of any point in time to a new cluster
- The recovery point object (RPO) is within several minutes.

For other backup needs, refer to Backup and Restore Overview to choose an appropriate backup method.

Note:

- Snapshot backup is only applicable to TiDB v3.1 or later releases.
- Log backup is only applicable to TiDB v6.3 or later releases.
- Data that is backed up using BR can only be restored to TiDB instead of other databases.

7.4.6.1.2 Ad-hoc backup

Ad-hoc backup includes snapshot backup and log backup. For log backup, you can start or stop a log backup task and clean log backup data.

To get an ad-hoc backup, you need to create a `Backup` Custom Resource (CR) object to describe the backup details. Then, 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 an example about how to back up the data of the `demo1` TiDB cluster in the `test1` Kubernetes namespace to Azure Blob Storage. The following are the detailed steps.

Prerequisites: Prepare an ad-hoc backup environment

1. Create a namespace for managing backup. The following example creates a `backup-test` namespace:

```
kubectl create namespace backup-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `backup-test` namespace:

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

3. Grant permissions to the remote storage for the `backup-test` namespace. You can grant permissions to Azure Blob Storage by two methods. For details, refer to Azure account permissions. After you grant the permissions, the `backup-test` namespace has a secret object named `azblob-secret` or `azblob-secret-ad`.

Note:

The role owned by the account must have the permission to modify blob at least (for example, a contributor).

When you create a secret object, you can use a customized name for the object. In this document, the name is `azblob-secret`.

4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.

    1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the backup database so that the `Backup` CR can adjust the GC time before and after the backup.
    2. Create `backup-demo1-tidb-secret` to store the account and password to access the TiDB cluster:

```
kubectl create secret generic backup-demo1-tidb-secret --from-
    ↪ literal=password=${password} --namespace=test1
```

Snapshot backup

To perform a snapshot backup, take the following steps:

Create the `Backup` CR named `demo1-full-backup-azblob` in the `backup-test` namespace:

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

The content of `full-backup-azblob.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-azblob
  namespace: backup-test
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
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-full-backup-folder
```

```
  #accessTier: Hot
```

When you configure `backup-azblob.yaml`, note the following:

- 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 Azure Blob Storage configuration, refer to Azure Blob Storage fields.
- Some parameters in `spec.br` are optional, such as `logLevel` and `statusAddr`. For more information about BR configuration, refer to BR fields.
- `spec.azblob.secretName`: fill in the name of the secret object, such as `azblob-secret` ↪ .
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure the `spec.tikvGCLifeTime` and `spec.from` fields in the `Backup` CR.
- For more information about the `Backup` CR fields, refer to Backup CR fields.

View the snapshot backup status

After you create the `Backup` CR, TiDB Operator starts the backup automatically. You can view the backup status by running the following command:

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

From the output, you can find the following information for the `Backup` CR named `demo1-full-backup-azblob`. The `COMMITTS` field indicates the time point of the snapshot backup:

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

Log backup

You can use a `Backup` CR to describe the start and stop of a log backup task and manage the log backup data. In this section, the example shows how to create a `Backup` CR named `demo1-log-backup-azblob`. See the following detailed steps.

Description of the `logSubcommand` field

In the Backup Custom Resource (CR), you can use the `logSubcommand` field to control the state of a log backup task. The `logSubcommand` field supports the following commands:

- `log-start`: initiates a new log backup task or resumes a paused task. Use this command to start the log backup process or resume a task from a paused state.

- `log-pause`: temporarily pauses the currently running log backup task. After pausing, you can use the `log-start` command to resume the task.
- `log-stop`: permanently stops the log backup task. After executing this command, the Backup CR enters a stopped state and cannot be restarted.

These commands provide fine-grained control over the lifecycle of log backup tasks, enabling you to start, pause, resume, and stop tasks effectively to manage log data retention in Kubernetes environments.

In TiDB Operator v1.5.4, v1.6.0, and earlier versions, you can use the `logStop: true` ↪ `/false` field to stop or start log backup tasks. This field is retained for backward compatibility.

However, do not use `logStop` and `logSubcommand` fields in the same Backup CR, as this is not supported. For TiDB Operator v1.5.5, v1.6.1, and later versions, it is recommended to use the `logSubcommand` field to ensure clear and consistent configuration.

Start log backup

1. In the `backup-test` namespace, create a `Backup` CR named `demo1-log-backup-azblob` ↪.

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

The content of `log-backup-azblob.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: backup-test
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
    #accessTier: Hot
```

2. Wait for the start operation to complete:

```
kubectl get jobs -n backup-test
```

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

3. View the newly created `Backup` CR:

```
kubectl get backup -n backup-test
```

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

View the log backup status

You can view the log backup status by checking the information of the `Backup` CR:

```
kubectl describe backup -n backup-test
```

From the output, you can find the following information for the `Backup` CR named `demo1 ↪ -log-backup-azblob`. The `Log Checkpoint Ts` field indicates the latest point in time that can be recovered:

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

Pause log backup

You can pause a log backup task by setting the `logSubcommand` field of the Backup Custom Resource (CR) to `log-pause`. The following example shows how to pause the `demo1-log-backup-azblob` CR created in Start log backup.

```
kubectl edit backup demo1-log-backup-azblob -n backup-test
```

To pause the log backup task, change the value of `logSubcommand` from `log-start` to `log-pause`, then save and exit the editor.

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

The modified content is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: backup-test
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
```

You can verify that the `STATUS` of the `demo1-log-backup-azblob` Backup CR changes from `Running` to `Pause`:

```
kubectl get backup -n backup-test
```

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

Resume log backup

If a log backup task is paused, you can resume it by setting the `logSubcommand` field to `log-start`. The following example shows how to resume the `demo1-log-backup-azblob` CR that was paused in Pause Log Backup.

> **Note:**
>
> This operation applies only to tasks in the `Pause` state. You cannot resume tasks in the `Fail` or `Stopped` state.

```
kubectl edit backup demo1-log-backup-azblob -n backup-test
```

To resume the log backup task, change the value of `logSubcommand` from `log-pause` to `log-start`, then save and exit the editor. The modified content is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: backup-test
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
    #accessTier: Hot
```

You can verify that the `STATUS` of the `demo1-log-backup-azblob` Backup CR changes from `Pause` to `Running`:

```
kubectl get backup -n backup-test
```

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

Stop log backup

You can stop a log backup task by setting the `logSubcommand` field of the Backup Custom Resource (CR) to `log-stop`. The following example shows how to stop the `demo1-log-backup-azblob` CR created in Start log backup.

```
kubectl edit backup demo1-log-backup-azblob -n backup-test
```

Change the value of `logSubcommand` to `log-stop`, then save and exit the editor. The modified content is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: backup-test
```

```
spec:
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
    #accessTier: Hot
```

You can verify that the `STATUS` of the `Backup` CR named `demo1-log-backup-azblob` changes from `Running` to `Stopped`:

```
kubectl get backup -n backup-test
```

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

`Stopped` is the terminal state for log backup. In this state, you cannot change the backup state again, but you can still clean up the log backup data.

In TiDB Operator v1.5.4, v1.6.0, and earlier versions, you can use the `logStop: true` ↪ `/false` field to stop or start log backup tasks. This field is retained for backward compatibility.

Clean log backup data

1. Because you already created a `Backup` CR named `demo1-log-backup-azblob` when you started log backup, you can clean the log data backup by modifying the same `Backup` CR. The following example shows how to clean log backup data generated before 2022-10-10T15:21:00+08:00.

   ```
   kubectl edit backup demo1-log-backup-azblob -n backup-test
   ```

   In the last line of the CR, append `spec.logTruncateUntil: "2022-10-10T15` ↪ `:21:00+08:00"`. Then save and quit the editor. The modified content is as follows:

   ```
   ---
   apiVersion: pingcap.com/v1alpha1
   kind: Backup
   metadata:
     name: demo1-backup-azblob
     namespace: backup-test
   ```

```
spec:
  backupMode: log
  logSubcommand: log-start/log-pause/log-stop
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
    #accessTier: Hot
  logTruncateUntil: "2022-10-10T15:21:00+08:00"
```

2. Wait for the clean operation to complete:

```
kubectl get jobs -n backup-test
```

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

3. View the `Backup` CR information:

```
kubectl describe backup -n backup-test
```

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

You can also view the information by running the following command:

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

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

Backup CR examples

Back up data of all clusters

```
---
apiVersion: pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
```

```
  namespace: backup-test
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    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-azblob
  namespace: backup-test
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    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-azblob
  namespace: backup-test
```

```
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    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-azblob
  namespace: backup-test
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  # ...
  br:
    cluster: demo1
    sendCredToTikv: false
    clusterNamespace: test1
  azblob:
    secretName: azblob-secret-ad
    container: my-container
    bucket: my-bucket
    prefix: my-folder
```

7.4.6.1.3 Scheduled snapshot 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 snapshot backup is

described by a custom `BackupSchedule` CR object. A snapshot backup is triggered at each backup time point. Its underlying implementation is the ad-hoc snapshot backup.

Prerequisites: Prepare a scheduled backup environment

Refer to Prepare an ad-hoc backup environment.

Perform a scheduled snapshot backup

Depending on which method you choose to grant permissions to the remote storage, perform a scheduled snapshot backup by doing one of the following:

- Method 1: If you grant permissions by access key, create the `BackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f backup-scheduler-azblob.yaml
```

The content of `backup-scheduler-azblob.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-azblob
  namespace: backup-test
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    br:
      cluster: demo1
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
      # sendCredToTikv: true
    azblob:
      secretName: azblob-secret-ad
      container: my-container
      prefix: my-folder
```

- Method 2: If you grant permissions by Azure AD, create the `BackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f backup-scheduler-azblob.yaml
```

The content of `backup-scheduler-azblob.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-azblob
  namespace: backup-test
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    br:
      cluster: demo1
      sendCredToTikv: false
      clusterNamespace: test1
      # logLevel: info
      # statusAddr: ${status_addr}
      # concurrency: 4
      # rateLimit: 0
      # timeAgo: ${time}
      # checksum: true
    azblob:
      secretName: azblob-secret-ad
      container: my-container
      prefix: my-folder
```

From the preceding content in `backup-scheduler-azblob.yaml`, you can see that the `backupSchedule` configuration consists of two parts. One is the unique configuration of `backupSchedule`, and the other is `backupTemplate`.

- For the unique configuration of `backupSchedule`, refer to BackupSchedule CR fields.
- `backupTemplate` specifies the configuration related to the cluster and remote storage, which is the same as the `spec` configuration of the `Backup` CR.

After creating the scheduled snapshot backup, you can run the following command to check the backup status:

```
kubectl get bks -n backup-test -o wide
```

You can run the following command to check all the backup items:

```
kubectl get backup -l tidb.pingcap.com/backup-schedule=demo1-backup-schedule
    ↪ -azblob -n backup-test
```

7.4.6.1.4 Integrated management of scheduled snapshot backup and log backup

You can use the `BackupSchedule` CR to integrate the management of scheduled snapshot backup and log backup for TiDB clusters. By setting the backup retention time, you can regularly recycle the scheduled snapshot backup and log backup, and ensure that you can perform PITR recovery through the scheduled snapshot backup and log backup within the retention period.

The following example creates a `BackupSchedule` CR named `integrated-backup-schedule-azblob`. For more information about the authorization method, refer to Azure account permissions.

Prerequisites: Prepare a scheduled snapshot backup environment

The steps to prepare for a scheduled snapshot backup are the same as those of Prepare for an ad-hoc backup.

Create `BackupSchedule`

1. Create a `BackupSchedule` CR named `integrated-backup-schedule-azblob` in the `backup-test` namespace.

   ```
   kubectl apply -f integrated-backup-scheduler-azblob.yaml
   ```

   The content of `integrated-backup-scheduler-azblob.yaml` is as follows:

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

```
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: schedule-backup-folder-snapshot
    #accessTier: Hot
logBackupTemplate:
  backupMode: log
  br:
    cluster: demo1
    clusterNamespace: test1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: schedule-backup-folder-log
    #accessTier: Hot
```

In the above example of `integrated-backup-scheduler-azblob.yaml`, the `backupSchedule` configuration consists of three parts: the unique configuration of `backupSchedule`, the configuration of the snapshot backup `backupTemplate`, and the configuration of the log backup `logBackupTemplate`.

For the field description of `backupSchedule`, refer to BackupSchedule CR fields.

2. After creating `backupSchedule`, use the following command to check the backup status:

```
kubectl get bks -n backup-test -o wide
```

A log backup task is created together with `backupSchedule`. You can check the log backup name through the `status.logBackup` field of the `backupSchedule` CR.

```
kubectl describe bks integrated-backup-schedule-azblob -n backup-test
```

3. To perform data restoration for a cluster, you need to specify the backup path. You can use the following command to check all the backup items under the scheduled snapshot backup.

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

The `MODE` field in the output indicates the backup mode. `snapshot` indicates the scheduled snapshot backup, and `log` indicates the log backup.

```
NAME                                                    MODE      STATUS
    ↪ ....
```

```
integrated-backup-schedule-azblob-2023-03-08t02-48-00 snapshot Complete
    ↪  ....
log-integrated-backup-schedule-azblob                log       Running
    ↪ ....
```

7.4.6.1.5 Delete the backup CR

If you no longer need the backup CR, you can delete it by referring to Delete the Backup CR.

7.4.6.1.6 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

7.4.6.2 Restore Data from Azure Blob Storage Using BR

This document describes how to restore the backup data stored in Azure Blob Storage to a TiDB cluster on Kubernetes, including two restoration methods:

- Full restoration. This method takes the backup data of snapshot backup and restores a TiDB cluster to the time point of the snapshot backup.
- Point-in-time recovery (PITR). This method takes the backup data of both snapshot backup and log backup and restores a TiDB cluster to any point in time.

The restore method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used to restore the data. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

PITR allows you to restore a new TiDB cluster to any point in time of the backup cluster. To use PITR, you need the backup data of snapshot backup and log backup. During the restoration, the snapshot backup data is first restored to the TiDB cluster, and then the log backup data between the snapshot backup time point and the specified point in time is restored to the TiDB cluster.

Note:

- BR is only applicable to TiDB v3.1 or later releases.
- PITR is only applicable to TiDB v6.3 or later releases.
- Data restored by BR cannot be replicated to a downstream cluster, because BR directly imports SST and LOG files to TiDB and the downstream cluster currently cannot access the upstream SST and LOG files.

7.4.6.2.1 Full restoration

This section provides an example about how to restore the backup data from the `spec.azblob.prefix` folder of the `spec.azblob.container` bucket on Azure Blob Storage to the `demo2` TiDB cluster in the `test2` namespace. The following are the detailed steps.

Prerequisites: Complete the snapshot backup

In this example, the `my-full-backup-folder` folder in the `my-container` bucket of Azure Blob Storage stores the snapshot backup data. For steps of performing snapshot backup, refer to Back up Data to Azure Blob Storage Using BR.

Step 1: Prepare the restoration environment

Before restoring backup data on Azure Blob Storage to TiDB using BR, take the following steps to prepare the restore environment:

1. Create a namespace for managing restoration. The following example creates a `restore-test` namespace:

```
kubectl create namespace restore-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `restore-test` namespace:

```
kubectl apply -f backup-rbac.yaml -n restore-test
```

3. Grant permissions to the remote storage for the `restore-test` namespace. You can grant permissions to Azure Blob Storage by two methods. For details, refer to Azure account permissions. After you grant the permissions, the `restore-test` namespace has a secret object named `azblob-secret` or `azblob-secret-ad`.

> **Note:**
>
> The role owned by the account must have the permission to modify blob at least (for example, a contributor).
>
> When you create a secret object, you can use a customized name for the object. In this document, the name is `azblob-secret`.

4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.

    1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the target database so that the `Restore` CR can adjust the GC time before and after the restore.
    2. Create the `restore-demo2-tidb-secret` secret to store the account and password to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-
    ↪ literal=password=${password} --namespace=test2
```

Step 2: Restore the backup data to a TiDB cluster

Create a `Restore` CR named `demo2-restore-azblob` in the `restore-test` namespace to restore cluster data as described below:

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

The content of `restore-full-azblob.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-azblob
  namespace: test2
spec:
  br:
    cluster: demo2
    clusterNamespace: test2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-folder
```

When configuring `restore-azblob.yaml`, note the following:

- For more information about Azure Blob Storage configuration, refer to Azure Blob Storage fields.
- Some parameters in `.spec.br` are optional, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about BR configuration, refer to BR fields.
- `spec.azblob.secretName`: fill in the name of the secret object, such as `azblob-secret` ↪ .
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure the `spec.to` fields in the `Restore` CR.

- For more information about the `Restore` CR fields, refer to Restore CR fields.

After creating the `Restore` CR, execute the following command to check the restore status:

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

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

7.4.6.2.2 Point-in-time recovery

This section provides an example about how to perform point-in-time recovery (PITR) in a `demo3` cluster in the `test3` namespace. PITR takes two steps:

1. Restore the cluster to the time point of the snapshot backup using the snapshot backup data in the `spec.pitrFullBackupStorageProvider.azblob.prefix` folder of the `spec.pitrFullBackupStorageProvider.azblob.container` bucket.
2. Restore the cluster to any point in time using the log backup data in the `spec.azblob.prefix` folder of the `spec.azblob.container` bucket.

The detailed steps are as follows.

Prerequisites: Complete data backup

In this example, the `my-container` bucket of Azure Blob Storage stores the following two types of backup data:

- The snapshot backup data generated during the **log backup**, stored in the `my-full-backup-folder-pitr` folder.
- The log backup data, stored in the `my-log-backup-folder-pitr` folder.

For detailed steps of how to perform data backup, refer to Back up data to Azure Blob Storage.

> **Note:**
>
> The specified restoration time point must be between the snapshot backup time point and the log backup `checkpoint-ts`.

Step 1: Prepare the restoration environment

Before restoring backup data on Azure Blob Storage to TiDB using BR, take the following steps to prepare the restoration environment:

1. Create a namespace for managing restoration. The following example creates a `restore-test` namespace:

```
kubectl create namespace restore-test
```

2. Download backup-rbac.yaml, and execute the following command to create the role-based access control (RBAC) resources in the `restore-test` namespace:

```
kubectl apply -f backup-rbac.yaml -n restore-test
```

3. Grant permissions to the remote storage for the `restore-test` namespace. You can grant permissions to Azure Blob Storage by two methods. For details, refer to Azure account permissions. After you grant the permissions, the `restore-test` namespace has a secret object named `azblob-secret` or `azblob-secret-ad`.

> **Note:**
>
> The role owned by the account must have the permission to access blob at least (for example, a reader).
>
> When you create a secret object, you can use a customized name for the object. In this document, the name is `azblob-secret`.

Step 2: Restore the backup data to a TiDB cluster

The example in this section restores the snapshot backup data to the cluster. The specified restoration time point must be between the time point of snapshot backup and the `Log Checkpoint Ts` of log backup.

The detailed steps are as follows:

1. Create a `Restore` CR named `demo3-restore-azblob` in the `restore-test` namespace and specify the restoration time point as `2022-10-10T17:21:00+08:00`:

```
kubectl apply -f restore-point-azblob.yaml
```

The content of `restore-point-azblob.yaml` is as follows:

```
---
apiVersion: pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-azblob
  namespace: restore-test
spec:
  restoreMode: pitr
  br:
```

```
  cluster: demo3
  clusterNamespace: test3
azblob:
  secretName: azblob-secret
  container: my-container
  prefix: my-log-backup-folder-pitr
pitrRestoredTs: "2022-10-10T17:21:00+08:00"
pitrFullBackupStorageProvider:
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-full-backup-folder-pitr
```

When you configure `restore-point-azblob.yaml`, note the following:

- `spec.restoreMode`: when you perform PITR, set this field to `pitr`. The default value of this field is `snapshot`, which means snapshot backup.

2. Wait for the restoration operation to complete:

```
kubectl get jobs -n restore-test
```

```
NAME                          COMPLETIONS  ...
restore-demo3-restore-azblob 1/1          ...
```

You can also check the restoration status by using the following command:

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

```
NAME                 STATUS    ...
demo3-restore-azblob Complete ...
```

7.4.6.2.3 Troubleshooting

If you encounter any problem during the restoration process, refer to Common Deployment Failures.

7.4.7 Persistent Volumes

7.4.7.1 Back up Data to PV

This document describes how to back up the data of a TiDB cluster on Kubernetes to Persistent Volumes (PVs). PVs in this documentation can be any Kubernetes supported PV types. This document uses NFS as an example PV type.

The backup method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used

to get the backup data of the TiDB cluster, and then send the data to PVs. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

7.4.7.1.1 Usage scenarios

If you have the following backup needs, you can use BR to make an ad-hoc backup or scheduled snapshot backup of the TiDB cluster data to PVs:

- To back up a large volume of data at a fast speed
- To get a direct backup of data as SST files (key-value pairs)

For other backup needs, refer to Backup and Restore Overview to choose an appropriate backup method.

> **Note:**
>
> - BR is only applicable to TiDB v3.1 or later releases.
> - Data that is backed up using BR can only be restored to TiDB instead of other databases.

7.4.7.1.2 Ad-hoc backup

Ad-hoc backup supports both snapshot backup and incremental backup.

To get an Ad-hoc backup, you need to create a `Backup` Custom Resource (CR) object to describe the backup details. Then, 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 an example about how to back up the data of the `demo1` TiDB cluster in the `test1` Kubernetes namespace to NFS. The following are the detailed steps.

Step 1: Prepare for an ad-hoc backup

1. Download backup-rbac.yaml to the server that runs the backup task.
2. 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
```

3. Make sure that the NFS server is accessible from your Kubernetes cluster, and you have configured TiKV 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
```

4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.

    1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the backup database so that the `Backup` CR can adjust the GC time before and after the backup.
    2. Create the `backup-demo1-tidb-secret` secret to store the account and password to access the TiDB cluster:

    ```
    kubectl create secret generic backup-demo1-tidb-secret --from-
        ↪ literal=password=${password} --namespace=test1
    ```

Step 2: Perform an 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
  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
```

When configuring `backup-nfs.yaml`, note the following:

- 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.
- `spec.local` refers to the configuration related to PVs. For more information about PV configuration, refer to Local storage fields.
- Some parameters in `spec.br` are optional, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, and `timeAgo`. For more information about `spec.br` fields, refer to BR fields.
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure `spec.tikvGCLifeTime` and `spec.from` fields in the `Backup` CR.
- For more information about the `Backup` CR fields, refer to Backup CR fields.

2. After creating the `Backup` CR, TiDB Operator automatically starts the backup task. You can 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
  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
  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
  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
  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
```

7.4.7.1.3 Scheduled snapshot 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 snapshot backup is described by a custom `BackupSchedule` CR object. A snapshot backup is triggered at each backup time point. Its underlying implementation is the ad-hoc snapshot backup.

Step 1: Prepare for a scheduled snapshot backup

The steps to prepare for a scheduled snapshot backup are the same as that of Prepare for an ad-hoc backup.

Step 2: Perform a scheduled snapshot 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:      br:          cluster: demo1        clusterNamesp
↪ : 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
```

```
From the `backup-schedule-nfs.yaml` 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`.

- For the unique configuration of `backupSchedule`, refer to [BackupSchedule
    ↪  CR fields](#backupschedule-cr-fields).
- `backupTemplate` specifies the configuration related to the cluster and
    ↪ remote storage, which is the same as the `spec` configuration of [the
    ↪  `Backup` CR](#backup-cr-fields).
```

2. After creating the scheduled snapshot 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
```

7.4.7.1.4 Delete the backup CR

If you no longer need the backup CR, refer to Delete the Backup CR.

7.4.7.1.5 Troubleshooting

If you encounter any problem during the backup process, refer to Common Deployment Failures.

7.4.7.2 Restore Data from PV

This document describes how to restore the TiDB cluster data backed up using TiDB Operator on Kubernetes. PVs in this documentation can be any Kubernetes supported PV types. This document shows how to restore data from NFS to TiDB.

The restore method described in this document is implemented based on CustomResourceDefinition (CRD) in TiDB Operator. For the underlying implementation, BR is used to restore the data. BR stands for Backup & Restore, which is a command-line tool for distributed backup and recovery of the TiDB cluster data.

7.4.7.2.1 Usage scenarios

After backing up TiDB cluster data to PVs using BR, if you need to recover the backup SST (key-value pairs) files from PVs to a TiDB cluster, you can follow steps in this document to restore the data using BR.

Note:

- BR is only applicable to TiDB v3.1 or later releases.
- Data restored by BR cannot be replicated to a downstream cluster, because BR directly imports SST files to TiDB and the downstream cluster currently cannot access the upstream SST files.

7.4.7.2.2 Step 1: Prepare the restore environment

Before restoring backup data on PVs to TiDB using BR, take the following steps to prepare the restore environment:

1. Download backup-rbac.yaml.
2. 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
```

3. Make sure that the NFS server is accessible from your Kubernetes cluster.
4. For a TiDB version earlier than v4.0.8, you also need to complete the following preparation steps. For TiDB v4.0.8 or a later version, skip these preparation steps.
    1. Make sure that you have the `SELECT` and `UPDATE` privileges on the `mysql.tidb` table of the target database so that the `Restore` CR can adjust the GC time before and after the restore.
    2. Create the `restore-demo2-tidb-secret` secret to store the account and password to access the TiDB cluster:

```
kubectl create secret generic restore-demo2-tidb-secret --from-
    ↪ literal=user=root --from-literal=password=<password> --
    ↪ namespace=test2
```

7.4.7.2.3 Step 2: Restore the backup data to a TiDB cluster

1. Create the `Restore` custom resource (CR), and restore the specified data to your cluster:

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

The content of the `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
  local:
    prefix: backup-nfs
    volume:
      name: nfs
      nfs:
        server: ${nfs_server_if}
        path: /nfs
    volumeMount:
      name: nfs
      mountPath: /nfs
```

When configuring `restore.yaml`, note the following:

- The example above restores data from the `local://${.spec.local.volume.nfs.path}/${.spec.local.prefix}/` directory on NFS to the `demo2` TiDB cluster in the `test2` namespace. For more information about PV configuration, refer to Local storage fields.
- Some parameters in `spec.br` are optional, such as `logLevel`, `statusAddr`, `concurrency`, `rateLimit`, `checksum`, `timeAgo`, and `sendCredToTikv`. For more information about `.spec.br`, refer to BR fields.
- For v4.0.8 or a later version, BR can automatically adjust `tikv_gc_life_time`. You do not need to configure the `spec.to` field in the `Restore` CR.
- 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:

```
kubectl get rt -n test2 -owide
```

7.4.7.2.4 Troubleshooting

If you encounter any problem during the restore process, refer to Common Deployment Failures.

7.4.8 Snapshot Backup and Restore across Multiple Kubernetes

7.4.8.1 BR Federation Architecture and Processes

BR Federation is a system designed to back up and restore TiDB clusters deployed across multiple Kubernetes using EBS snapshots.

Normally, TiDB Operator can only access the Kubernetes cluster where it is deployed. This means a TiDB Operator can only back up TiKV volumes' snapshots within its own Kubernetes cluster. However, to perform EBS snapshot backup and restore across multiple Kubernetes clusters, a coordinator role is required. This is where the BR Federation comes in.

This document outlines the architecture of the BR Federation and the processes involved in backup and restoration.

7.4.8.1.1 BR Federation architecture

BR Federation operates as the control plane, interacting with the data plane, which includes each Kubernetes cluster where TiDB components are deployed. The interaction is facilitated through the Kubernetes API Server.

BR Federation coordinates `Backup` and `Restore` Custom Resources (CRs) in the data plane to accomplish backup and restoration across multiple Kubernetes clusters.

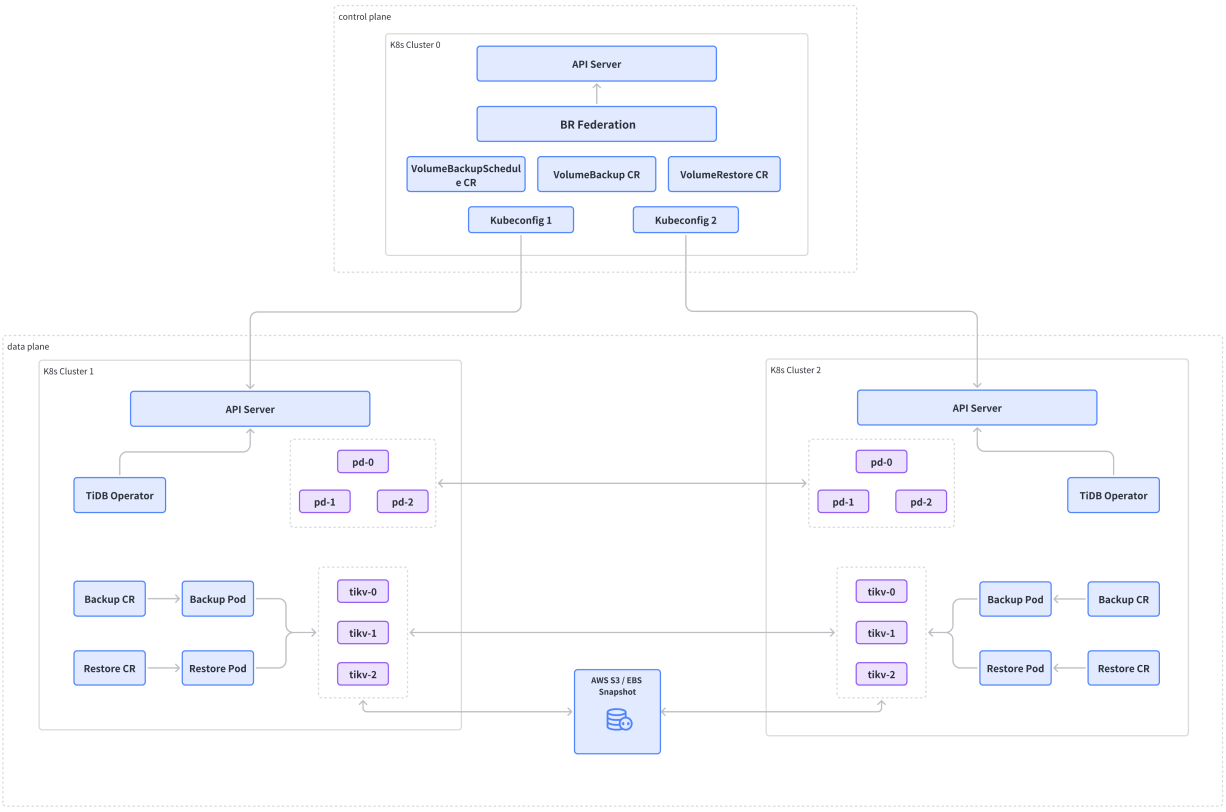


Figure 5: BR Federation architecture

7.4.8.1.2 Backup process

Backup process in data plane

The backup process in the data plane consists of three phases:

1. **Phase One:** TiDB Operator schedules a backup pod to request PD to pause region scheduling and Garbage Collection (GC). As each TiKV instance might take snapshots at different times, pausing scheduling and GC can avoid data inconsistencies between TiKV instances during snapshot taking. Since the TiDB components are interconnected across multiple Kubernetes clusters, executing this operation in one Kubernetes cluster affects the entire TiDB cluster.

2. **Phase Two:** TiDB Operator collects meta information such as `TidbCluster` CR and EBS volumes, and then schedules another backup pod to request AWS API to create EBS snapshots. This phase must be executed in each Kubernetes cluster.

3. **Phase Three:** After EBS snapshots are completed, TiDB Operator deletes the first backup pod to resume region scheduling and GC for the TiDB cluster. This operation is required only in the Kubernetes cluster where Phase One was executed.

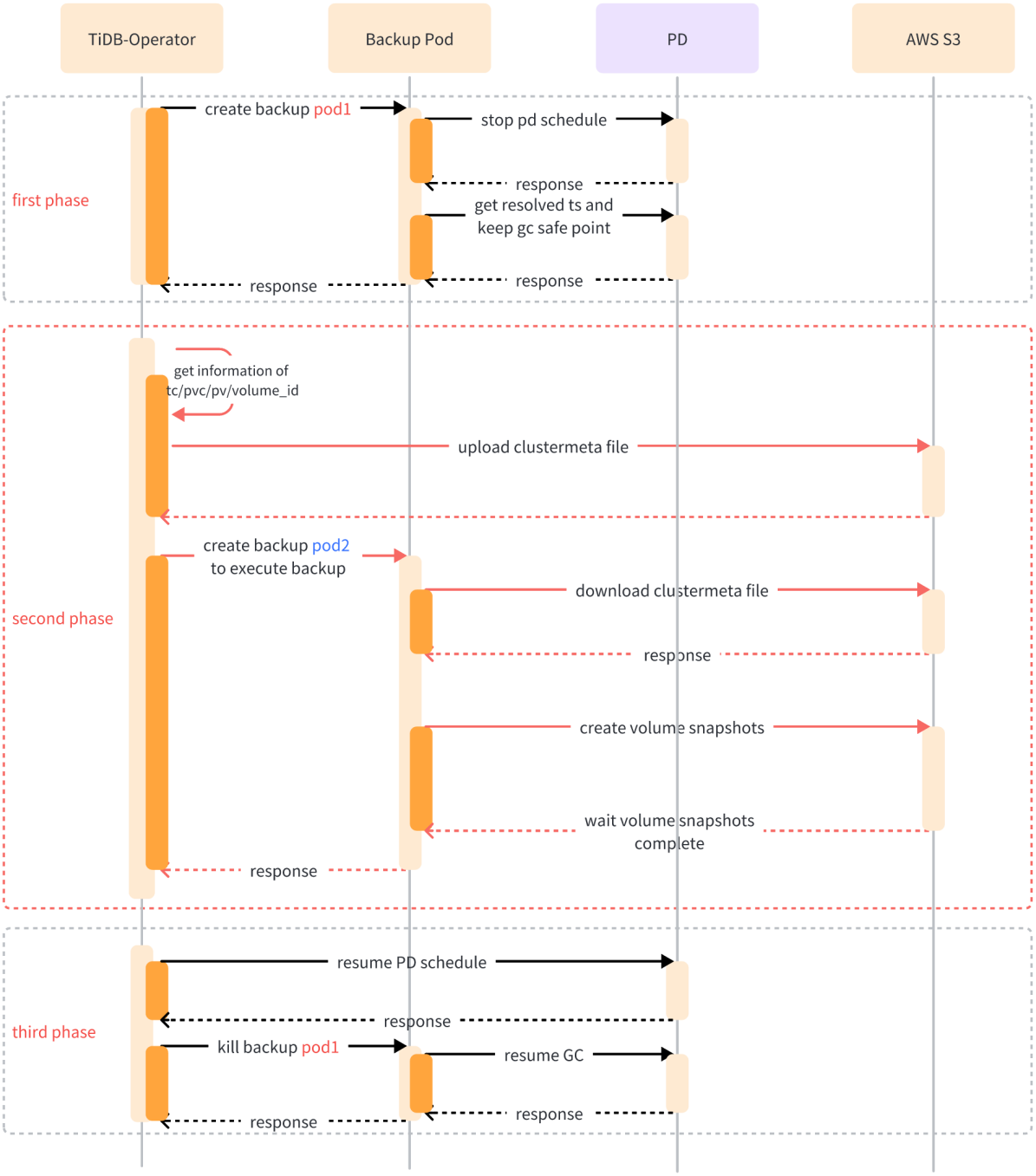


Figure 6: backup process in data plane

Backup orchestration process

The orchestration process of `Backup` from the control plane to the data plane is as follows:

Figure 7: backup orchestration process

7.4.8.1.3 Restore process

Restore process in data plane

The restore process in the data plane consists of three phases:

1. **Phase One:** TiDB Operator schedules a restore pod to request the AWS API to restore the EBS volumes using EBS snapshots based on the backup information. The

volumes are then mounted onto the TiKV nodes, and TiKV instances are started in recovery mode. This phase must be executed in each Kubernetes cluster.

2. **Phase Two:** TiDB Operator schedules another restore pod to restore all raft logs and KV data in TiKV instances to a consistent state, and then instructs TiKV instances to exit recovery mode. As TiKV instances are interconnected across multiple Kubernetes clusters, this operation can restore all TiKV data and only needs to be executed in one Kubernetes cluster.

3. **Phase Three:** TiDB Operator restarts all TiKV instances to run in normal mode, and start TiDB finally. This phase must be executed in each Kubernetes cluster.

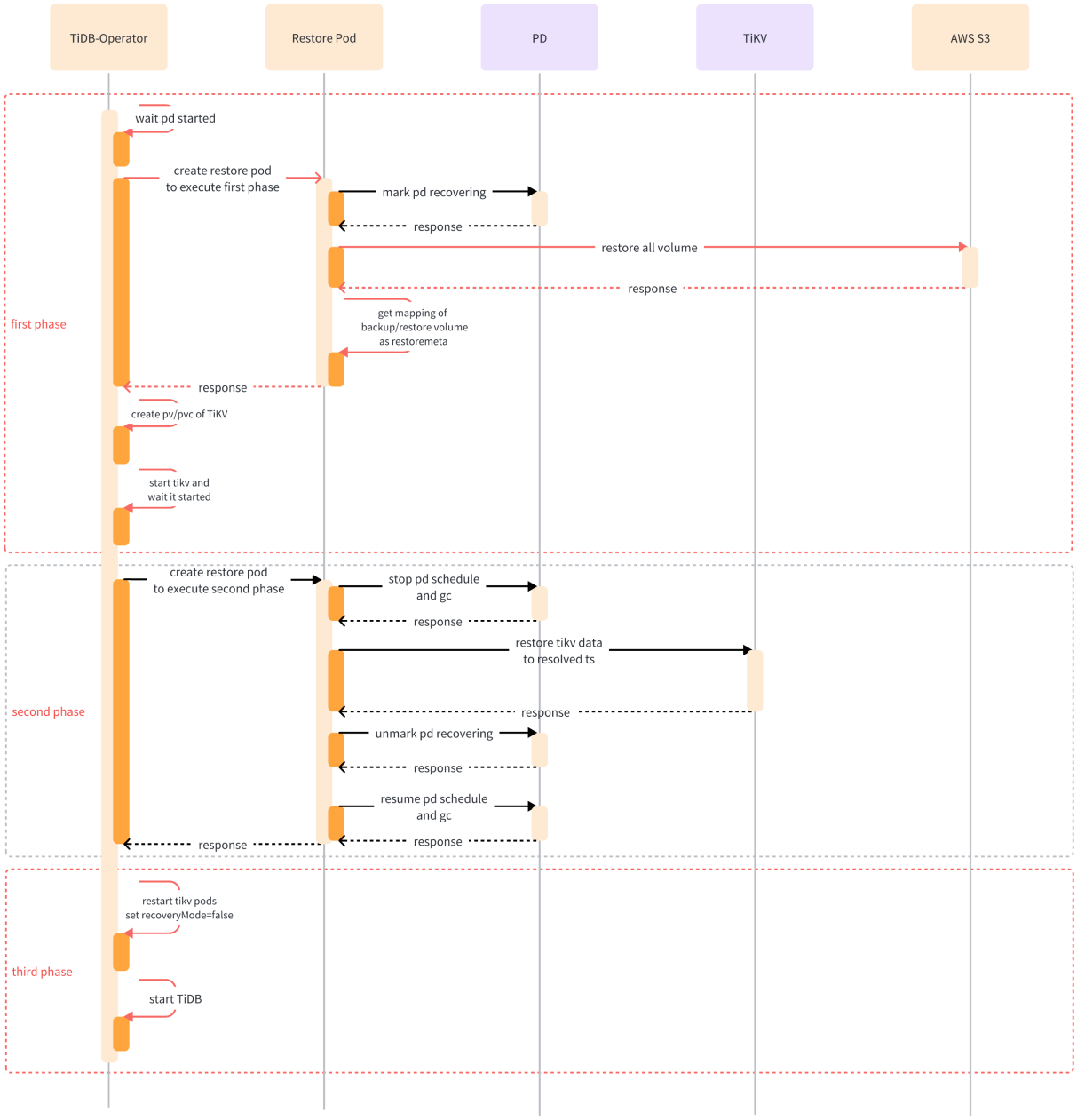


Figure 8: restore process in data plane

Restore orchestration process

The orchestration process of `Restore` from the control plane to the data plane is as follows:

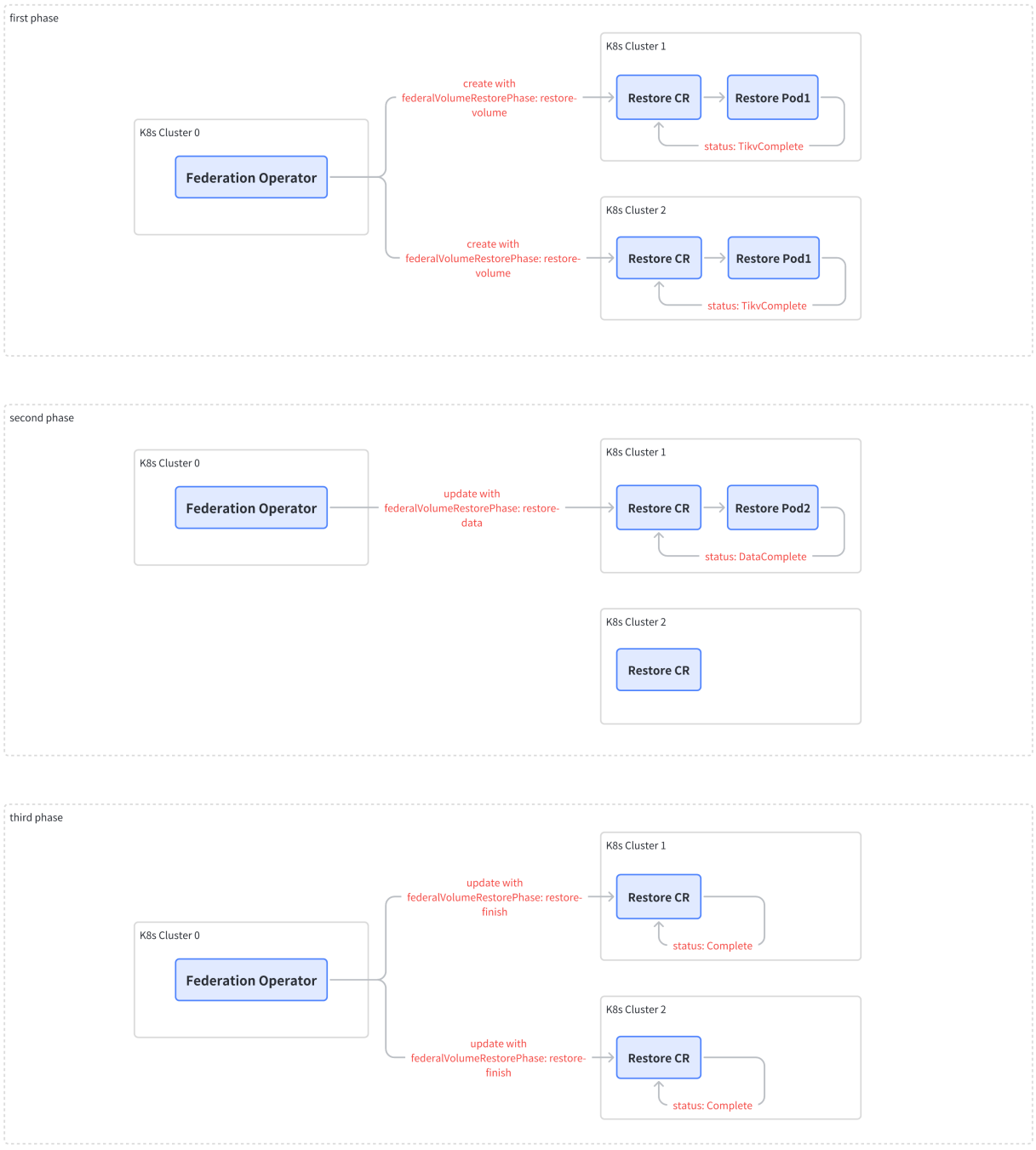


Figure 9: restore orchestration process

7.4.8.2 Deploy BR Federation on Kubernetes

This document describes how to deploy BR Federation across multiple Kubernetes clusters.

7.4.8.2.1 Prerequisites

Before deploy BR Federation on Kubernetes cluster, make sure you have met the following prerequisites:

- Kubernetes version must be >= v1.24.
- You must have multiple Kubernetes clusters.
- You have deployed TiDB Operator for all the Kubernetes clusters that serve as data planes.

7.4.8.2.2 Step 1: Generate a kubeconfig file in data planes

The BR Federation manages Kubernetes clusters of data planes by accessing their API servers. To authenticate and authorize itself in the API servers, BR Federation requires a kubeconfig file. The users or service accounts in the kubeconfig file need to have at least all the permissions of **backups.pingcap.com** and **restores.pingcap.com** CRD.

You can get the kubeconfig file from the Kubernetes cluster administrator. However, if you have permission to access all the data planes, you can generate the kubeconfig file on your own.

Step 1.1: Create RBAC resources in data planes

To enable the BR Federation to manipulate Backup and Restore CR, you need to create the following resources in every data plane.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: br-federation-member
  namespace: tidb-admin
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: br-federation-manager:br-federation-member
rules:
- apiGroups:
  - pingcap.com
  resources:
  - backups
  - restores
  verbs:
  - '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
```

```
  name: br-federation-manager:br-federation-member
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: br-federation-manager:br-federation-member
subjects:
- kind: ServiceAccount
  name: br-federation-member
  namespace: tidb-admin
```

For Kubernetes >= v1.24, to let external applications access the Kubernetes API server, you need to manually create a service account secret as follows:

```
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
 name: br-federation-member-secret
 namespace: tidb-admin
 annotations:
   kubernetes.io/service-account.name: "br-federation-member"
```

Step 1.2: Generate kubeconfig files

Execute the following script for every data plane.

```
#### for Kubernetes < 1.24
export TOKEN_SECRET_NAME=$(kubectl -n tidb-admin get serviceaccount br-
    ↪ federation-member -o=jsonpath='{.secrets[0].name}')
#### for Kubernetes >= 1.24, the service account secret should be created
    ↪ manually as above, so you should use its name as value of
    ↪ TOKEN_SECRET_NAME
#### export TOKEN_SECRET_NAME=br-federation-member-secret
export USER_TOKEN_VALUE=$(kubectl -n tidb-admin get secret/${
    ↪ TOKEN_SECRET_NAME} -o=go-template='{{.data.token}}' | base64 --decode
    ↪ )
export CURRENT_CONTEXT=$(kubectl config current-context)
export CURRENT_CLUSTER=$(kubectl config view --raw -o=go-template='{{range .
    ↪ contexts}}{{if eq .name "'''${CURRENT_CONTEXT}'''"}}{{ index .context
    ↪  "cluster" }}{{end}}{{end}}')
export CLUSTER_CA=$(kubectl config view --raw -o=go-template='{{range .
    ↪ clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}"{{with index .
    ↪ cluster "certificate-authority-data" }}{{.}}{{end}}"{{ end }}{{ end
    ↪ }}')
export CLUSTER_SERVER=$(kubectl config view --raw -o=go-template='{{range .
    ↪ clusters}}{{if eq .name "'''${CURRENT_CLUSTER}'''"}}{{ .cluster.
    ↪ server }}{{end}}{{ end }}')
```

```
#### you should modify this value in different data plane
export DATA_PLANE_SYMBOL="a"

cat << EOF > {k8s-name}-kubeconfig
apiVersion: v1
kind: Config
current-context: ${DATA_PLANE_SYMBOL}
contexts:
- name: ${DATA_PLANE_SYMBOL}
  context:
    cluster: ${CURRENT_CLUSTER}
    user: br-federation-member-${DATA_PLANE_SYMBOL}
    namespace: kube-system
clusters:
- name: ${CURRENT_CLUSTER}
  cluster:
    certificate-authority-data: ${CLUSTER_CA}
    server: ${CLUSTER_SERVER}
users:
- name: br-federation-member-${DATA_PLANE_SYMBOL}
  user:
    token: ${USER_TOKEN_VALUE}
EOF
```

The environment variable `$DATA_PLANE_SYMBOL` represents the name of the data plane cluster. Make sure that you provide a brief and unique name. In the preceding script, you use this variable as the context name for kubeconfig. The context name will be used as `k8sClusterName` in both the `VolumeBackup` and `VolumeRestore` CR.

Step 1.3: Merge multiple kubeconfig files into one

After following the previous steps to generate kubeconfig, you now have multiple kubeconfig files. You need to merge them into a single kubeconfig file.

Assume that you have 3 kubeconfig files with file paths: `kubeconfig-path1`, `kubeconfig-path2`, `kubeconfig-path3`. To merge these files into one kubeconfig file with file path `data-planes-kubeconfig`, execute the following command:

```
KUBECONFIG=${kubeconfig-path1}:${kubeconfig-path2}:${kubeconfig-path3}
    ↪ kubectl config view --flatten > ${data-planes-kubeconfig}
```

7.4.8.2.3 Step 2: Deploy BR Federation in the control plane

To deploy the BR Federation, you need to select one Kubernetes cluster as the control plane. The following steps **must be executed on the control plane**.

Step 2.1: Create CRD

The BR Federation uses Custom Resource Definition (CRD) to extend Kubernetes. Before using the BR Federation, you must create the CRD in your Kubernetes cluster. After using the BR Federation Manager, you only need to perform the operation once.

```
kubectl create -f https://raw.githubusercontent.com/pingcap/tidb-operator/v1
    ↪ .6.1/manifests/federation-crd.yaml
```

Step 2.2: Prepare the kubeconfig secret

Now that you already have a kubeconfig file of data planes, you need to encode the kubeconfig file into a secret. Take the following steps:

1. Encode the kubeconfig file:

   ```
   base64 -i ${kubeconfig-path}
   ```

2. Store the output from the previous step in a secret object.

   Note that the name of the secret and the data key of the kubeconfig field **must** match the following example:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: br-federation-kubeconfig
   type: Opaque
   data:
     kubeconfig: ${encoded-kubeconfig}
   ```

Step 2.3: Install BR Federation

This section describes how to install the BR Federation using Helm 3.

- If you prefer to use the default configuration, follow the **Quick deployment** steps.
- If you prefer to use a custom configuration, follow the **Custom deployment** steps.

1. To create resources related to the BR Federation, create a namespace:

   ```
   kubectl create ns br-fed-admin
   ```

2. In the specified namespace, create a secret that contains all the encoded kubeconfig files:

   ```
   kubectl create -f ${secret-path} -n br-fed-admin
   ```

3. Add the PingCAP repository:

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

4. Install the BR Federation:

```
helm install --namespace br-fed-admin br-federation pingcap/br-
    ↪ federation --version v1.6.1
```

1. To create resources related to the BR Federation, create a namespace:

```
kubectl create ns br-fed-admin
```

2. In the specified namespace, create a secret that contains all the encoded kubeconfig files:

```
kubectl create -f ${secret-path} -n br-fed-admin
```

3. Add the PingCAP repository:

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

4. Get the `values.yaml` file of the desired `br-federation` chart for deployment.

```
mkdir -p ${HOME}/br-federation && \
helm inspect values pingcap/br-federation --version=v1.6.1 > ${HOME}/br
    ↪ -federation/values.yaml
```

5. Configure the BR Federation by modifying fields such as `image`, `limits`, `requests`, and `replicas` according to your needs.

6. Deploy the BR Federation.

```
helm install --namespace br-fed-admin br-federation pingcap/br-
    ↪ federation --version v1.6.1 -f ${HOME}/br-federation/values.yaml
    ↪ && \
kubectl get po -n br-fed-admin -l app.kubernetes.io/instance=br-
    ↪ federation
```

7.4.8.2.4 What's next

After deploying BR Federation, you can now perform the following tasks:

- Back Up a TiDB Cluster across Multiple Kubernetes Using EBS Volume Snapshots
- Restore a TiDB Cluster across Multiple Kubernetes from EBS Volume Snapshots

7.4.8.3 Back Up a TiDB Cluster across Multiple Kubernetes Using EBS Volume Snapshots

This document describes how to back up the data of a TiDB cluster deployed across multiple AWS Kubernetes clusters to AWS storage using EBS volume snapshots.

The backup method described in this document is implemented based on CustomResourceDefinition (CRD) in BR Federation and TiDB Operator. BR (Backup & Restore) is a command-line tool for distributed backup and recovery of the TiDB cluster data. For the underlying implementation, BR gets the backup data of the TiDB cluster, and then sends the data to the AWS storage.

> **Note**
>
> Before you back up data, make sure that you have deployed BR Federation.

7.4.8.3.1 Usage scenarios

If you have the following requirements when backing up TiDB cluster data, you can use TiDB Operator to back up the data using volume snapshots and metadata to Amazon S3:

- Minimize the impact of backup, such as keeping the impact on QPS and transaction latency less than 5%, and not utilizing cluster CPU and memory.
- Back up and restore data in a short period of time. For example, completing a backup within 1 hour and restore it within 2 hours.

If you have any other requirements, refer to Backup and Restore Overview and select an appropriate backup method.

7.4.8.3.2 Prerequisites

Storage blocks on volumes that were created from snapshots must be initialized (pulled down from Amazon S3 and written to the volume) before you can access the block. This preliminary action takes time and can cause a significant increase in the latency of an I/O operation the first time each block is accessed. Volume performance is achieved after all blocks have been downloaded and written to the volume.

According to AWS documentation, the EBS volume restored from snapshots might have high latency before it is initialized. This can impact the performance of a restored TiDB cluster. See details in Create a volume from a snapshot.

To initialize the restored volume more efficiently, it is recommended to **separate WAL and raft log into a dedicated small volume apart from TiKV data**. By fully initializing the volume of WAL and raft log separately, we can enhance write performance for a restored TiDB cluster.

7.4.8.3.3 Limitations

- Snapshot backup is applicable to TiDB Operator v1.5.1 or later versions, and TiDB v6.5.4 or later versions.
- For TiKV configuration, do not set `resolved-ts.enable` to `false`, and do not set `raftstore.report-min-resolved-ts-interval` to `"0s"`. Otherwise, it can lead to backup failure.
- For PD configuration, do not set `pd-server.min-resolved-ts-persistence-interval` to `"0s"`. Otherwise, it can lead to backup failure.
- To use this backup method, the TiDB cluster must be deployed on AWS EC2 and use AWS EBS volumes.
- This backup method is currently not supported for TiFlash, TiCDC, DM, and TiDB Binlog nodes.

Note:

- To perform volume snapshot restore, ensure that the TiKV configuration during restore is consistent with the configuration used during backup.
  - To check consistency, download the `backupmeta` file from the backup file stored in Amazon S3, and check the `kubernetes.crd_tidb_cluster.spec` field.
  - If this field is inconsistent, you can modify the TiKV configuration by referring to Configure a TiDB Cluster on Kubernetes.
- If Encryption at Rest is enabled for TiKV KMS, ensure that the master key is enabled for AWS KMS during restore.

7.4.8.3.4 Ad-hoc backup

You can either fully or incrementally back up snapshots based on AWS EBS volumes. The initial backup of a node is full backup, while subsequent backups are incremental backup.

Snapshot backup is defined in a customized `VolumeBackup` custom resource (CR) object. The BR Federation completes the backup task according to the specifications in this object.

Step 1. Set up the environment for EBS volume snapshot backup in every data plane

You must execute the following steps in every data plane.

1. Download the `backup-rbac.yaml` file to the backup server.
2. If you have deployed the TiDB cluster in `${namespace}`, create the RBAC-related resources required for the backup in this namespace by running the following command:

```
kubectl apply -f backup-rbac.yaml -n ${namespace}
```

3. Grant permissions to access remote storage.

   To back up cluster data and save snapshot metadata to Amazon S3, you need to grant permissions to remote storage. Refer to AWS account authorization for the three available methods.

Step 2. Back up data to S3 storage

You must execute the following steps in the control plane.

Depending on the authorization method you choose in the previous step for granting remote storage access, you can back up data by EBS snapshots using any of the following methods accordingly:

If you grant permissions by accessKey and secretKey, you can create the `VolumeBackup` CR as follows:

```
kubectl apply -f backup-fed.yaml
```

The `backup-fed.yaml` file has the following content:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeBackup
metadata:
  name: ${backup-name}
spec:
  clusters:
  - k8sClusterName: ${k8s-name1}
    tcName: ${tc-name1}
    tcNamespace: ${tc-namespace1}
  - k8sClusterName: ${k8s-name2}
    tcName: ${tc-name2}
    tcNamespace: ${tc-namespace2}
  - ... # other clusters
  template:
    br:
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: ${region-name}
      bucket: ${bucket-name}
      prefix: ${backup-path}
    toolImage: ${br-image}
```

```
    cleanPolicy: Delete
    calcSizeLevel: {snapshot-size-calculation-level}
```

If you grant permissions by associating Pod with IAM, you can create the `VolumeBackup` CR as follows:

```
kubectl apply -f backup-fed.yaml
```

The `backup-fed.yaml` file has the following content:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeBackup
metadata:
  name: ${backup-name}
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/role-name
spec:
  clusters:
  - k8sClusterName: ${k8s-name1}
    tcName: ${tc-name1}
    tcNamespace: ${tc-namespace1}
  - k8sClusterName: ${k8s-name2}
    tcName: ${tc-name2}
    tcNamespace: ${tc-namespace2}
  - ... # other clusters
  template:
    br:
      sendCredToTikv: false
    s3:
      provider: aws
      region: ${region-name}
      bucket: ${bucket-name}
      prefix: ${backup-path}
    toolImage: ${br-image}
    cleanPolicy: Delete
    calcSizeLevel: {snapshot-size-calculation-level}
```

If you grant permissions by associating ServiceAccount with IAM, you can create the `VolumeBackup` CR as follows:

```
kubectl apply -f backup-fed.yaml
```

The `backup-fed.yaml` file has the following content:

```
---
apiVersion: federation.pingcap.com/v1alpha1
```

```
kind: VolumeBackup
metadata:
  name: ${backup-name}
spec:
  clusters:
  - k8sClusterName: ${k8s-name1}
    tcName: ${tc-name1}
    tcNamespace: ${tc-namespace1}
  - k8sClusterName: ${k8s-name2}
    tcName: ${tc-name2}
    tcNamespace: ${tc-namespace2}
  - ... # other clusters
  template:
    br:
      sendCredToTikv: false
    s3:
      provider: aws
      region: ${region-name}
      bucket: ${bucket-name}
      prefix: ${backup-path}
    toolImage: ${br-image}
    serviceAccount: tidb-backup-manager
    cleanPolicy: Delete
    calcSizeLevel: {snapshot-size-calculation-level}
```

Note:

The value of `spec.clusters.k8sClusterName` field in `VolumeBackup` CR must be the same as the **context name** of the kubeconfig used by the br-federation-manager.

Step 3. View the backup status

After creating the `VolumeBackup` CR, the BR Federation automatically starts the backup process in each data plane.

To check the volume backup status, use the following command:

```
kubectl get vbk -n ${namespace} -o wide
```

Once the volume backup is complete, you can get the information of all the data planes in the `status.backups` field. This information can be used for volume restore.

To obtain the information, use the following command:

```
kubectl get vbk ${backup-name} -n ${namespace} -o yaml
```

The information is as follows:

```
status:
  backups:
  - backupName: fed-{backup-name}-{k8s-name1}
    backupPath: s3://{bucket-name}/{backup-path}-{k8s-name1}
    commitTs: "ts1"
    k8sClusterName: {k8s-name1}
    tcName: {tc-name1}
    tcNamespace: {tc-namespace1}
  - backupName: fed-{backup-name}-{k8s-name2}
    backupPath: s3://{bucket-name}/{backup-path}-{k8s-name2}
    commitTs: "ts2"
    k8sClusterName: {k8s-name2}
    tcName: {tc-name2}
    tcNamespace: {tc-namespace2}
  - ... # other backups
```

Delete the `VolumeBackup` CR

If you set `spec.template.cleanPolicy` to `Delete`, when you delete the `VolumeBackup` CR, the BR Federation will clean up the backup file and the volume snapshots on AWS.

To delete the `VolumeBackup` CR, run the following commands:

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

7.4.8.3.5 Scheduled volume backup

To ensure regular backups of the TiDB cluster and prevent an excessive number of backup items, you can set a backup policy and retention policy.

This can be done by creating a `VolumeBackupSchedule` CR object that describes the scheduled snapshot backup. Each backup time point triggers a volume backup. The underlying implementation is the ad-hoc volume backup.

Perform a scheduled volume backup

You must execute the following steps in the control plane.

Depending on the authorization method you choose in the previous step for granting remote storage access, perform a scheduled volume backup by doing one of the following:

If you grant permissions by accessKey and secretKey, Create the `VolumeBackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f volume-backup-scheduler.yaml
```

The content of `volume-backup-scheduler.yaml` is as follows:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeBackupSchedule
metadata:
  name: {scheduler-name}
  namespace: {namespace-name}
spec:
  #maxBackups: {number}
  #pause: {bool}
  maxReservedTime: {duration}
  schedule: {cron-expression}
  backupTemplate:
    clusters:
      - k8sClusterName: {k8s-name1}
        tcName: {tc-name1}
        tcNamespace: {tc-namespace1}
      - k8sClusterName: {k8s-name2}
        tcName: {tc-name2}
        tcNamespace: {tc-namespace2}
      - ... # other clusters
    template:
      br:
        sendCredToTikv: true
      s3:
        provider: aws
        secretName: s3-secret
        region: {region-name}
        bucket: {bucket-name}
        prefix: {backup-path}
      toolImage: {br-image}
      cleanPolicy: Delete
      calcSizeLevel: {snapshot-size-calculation-level}
```

If you grant permissions by associating Pod with IAM, Create the `VolumeBackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f volume-backup-scheduler.yaml
```

The content of `volume-backup-scheduler.yaml` is as follows:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeBackupSchedule
metadata:
```

```
  name: {scheduler-name}
  namespace: {namespace-name}
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/role-name
spec:
  #maxBackups: {number}
  #pause: {bool}
  maxReservedTime: {duration}
  schedule: {cron-expression}
  backupTemplate:
    clusters:
      - k8sClusterName: {k8s-name1}
        tcName: {tc-name1}
        tcNamespace: {tc-namespace1}
      - k8sClusterName: {k8s-name2}
        tcName: {tc-name2}
        tcNamespace: {tc-namespace2}
      - ... # other clusters
    template:
      br:
        sendCredToTikv: false
      s3:
        provider: aws
        region: {region-name}
        bucket: {bucket-name}
        prefix: {backup-path}
      toolImage: {br-image}
      cleanPolicy: Delete
      calcSizeLevel: {snapshot-size-calculation-level}
```

If you grant permissions by associating ServiceAccount with IAM, Create the `VolumeBackupSchedule` CR, and back up cluster data as described below:

```
kubectl apply -f volume-backup-scheduler.yaml
```

The content of `volume-backup-scheduler.yaml` is as follows:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeBackupSchedule
metadata:
  name: {scheduler-name}
  namespace: {namespace-name}
spec:
  #maxBackups: {number}
  #pause: {bool}
```

```
maxReservedTime: {duration}
schedule: {cron-expression}
backupTemplate:
  clusters:
    - k8sClusterName: {k8s-name1}
      tcName: {tc-name1}
      tcNamespace: {tc-namespace1}
    - k8sClusterName: {k8s-name2}
      tcName: {tc-name2}
      tcNamespace: {tc-namespace2}
    - ... # other clusters
  template:
    br:
      sendCredToTikv: false
    s3:
      provider: aws
      region: {region-name}
      bucket: {bucket-name}
      prefix: {backup-path}
    serviceAccount: tidb-backup-manager
    toolImage: {br-image}
    cleanPolicy: Delete
    calcSizeLevel: {snapshot-size-calculation-level}
```

7.4.8.4 Restore a TiDB Cluster across Multiple Kubernetes from EBS Volume Snapshots

This document describes how to restore backup data in AWS EBS snapshots to a TiDB cluster across multiple Kubernetes clusters.

The restore method described in this document is implemented based on CustomResourceDefinition (CRD) in BR Federation and TiDB Operator. BR (Backup & Restore) is a command-line tool for distributed backup and recovery of the TiDB cluster data. For the underlying implementation, BR restores the data.

> **Note**
>
> Before you restore data, make sure that you have deployed BR Federation.

7.4.8.4.1 Limitations

- Snapshot restore is applicable to TiDB Operator v1.5.1 or later versions and TiDB v6.5.4 or later versions.

- You can use snapshot restore only to restore data to a cluster with the same number of TiKV nodes and volumes configuration. That is, the number of TiKV nodes and volume configurations of TiKV nodes are identical between the restore cluster and backup cluster.
- Snapshot restore is currently not supported for TiFlash, TiCDC, DM, and TiDB Binlog nodes.

7.4.8.4.2 Prerequisites

Before restoring a TiDB cluster across multiple Kubernetes clusters from EBS volume snapshots, you need to complete the following preparations.

- Complete the volume backup

  For detailed steps, refer to Back Up a TiDB Cluster across Multiple Kubernetes Using EBS Volume Snapshots.

- Prepare the restore cluster

  - Deploy a TiDB cluster across multiple Kubernetes clusters that you want to restore data to. For detailed steps, refer to Deploy a TiDB Cluster across Multiple Kubernetes Clusters.
  - When deploying the TiDB cluster, add the `recoveryMode: true` field to the spec of `TidbCluster`.

> **Note:**
>
> The EBS volume restored from snapshots might have high latency before it is initialized. This can impact the performance of a restored TiDB cluster. See details in Create a volume from a snapshot.
>
> It is recommended that you configure `spec.template.warmup: sync` to initialize TiKV volumes automatically during the restoration process.

7.4.8.4.3 Restore process

Step 1. Set up the environment for EBS volume snapshot restore in every data plane

You must execute the following steps in every data plane.

1. Download the backup-rbac.yaml file to the restore server.
2. Create the RBAC-related resources required for the restore by running the following command. Note that the RBAC-related resources must be put in the same `${namespace}` as the TiDB cluster.

```
kubectl apply -f backup-rbac.yaml -n ${namespace}
```

3. Grant permissions to access remote storage.

   To restore data from EBS snapshots, you need to grant permissions to remote storage. Three ways are available. Refer to AWS account authorization for the three available methods.

Step 2. Restore data to the TiDB cluster

You must execute the following steps in the control plane.

Depending on the authorization method you choose in the previous step for granting remote storage access, you can restore data to TiDB using any of the following methods accordingly:

> **Note:**
>
> Snapshot restore creates volumes with the default configuration (3000 IOPS/125 MB/s) of GP3. To perform restore using other configurations, you can specify the volume type or configuration, such as `--volume-type=gp3`, `--volume-iops=7000`, or `--volume-throughput=400`, and they are shown in the following examples.

If you grant permissions by accessKey and secretKey, you can create the `VolumeRestore` CR as follows:

```
kubectl apply -f restore-fed.yaml
```

The `restore-fed.yaml` file has the following content:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeRestore
metadata:
  name: ${restore-name}
spec:
  clusters:
  - k8sClusterName: ${k8s-name1}
    tcName: ${tc-name1}
    tcNamespace: ${tc-namespace1}
    backup:
      s3:
        provider: aws
```

```
      secretName: s3-secret
      region: ${region-name}
      bucket: ${bucket-name}
      prefix: ${backup-path1}
  - k8sClusterName: ${k8s-name2}
    tcName: ${tc-name2}
    tcNamespace: ${tc-namespace2}
    backup:
      s3:
        provider: aws
        secretName: s3-secret
        region: ${region-name}
        bucket: ${bucket-name}
        prefix: ${backup-path2}
  - ... # other clusters
  template:
    br:
      sendCredToTikv: true
      options:
      - --volume-type=gp3
      - --volume-iops=7000
      - --volume-throughput=400
    toolImage: ${br-image}
    warmup: sync
    warmupImage: ${wamrup-image}
```

If you grant permissions by associating Pod with IAM, you can create the `VolumeRestore` CR as follows:

```
kubectl apply -f restore-fed.yaml
```

The `restore-fed.yaml` file has the following content:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeRestore
metadata:
  name: ${restore-name}
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/role-name
spec:
  clusters:
  - k8sClusterName: ${k8s-name1}
    tcName: ${tc-name1}
    tcNamespace: ${tc-namespace1}
    backup:
```

```
    s3:
      provider: aws
      region: ${region-name}
      bucket: ${bucket-name}
      prefix: ${backup-path1}
  - k8sClusterName: ${k8s-name2}
    tcName: ${tc-name2}
    tcNamespace: ${tc-namespace2}
    backup:
      s3:
        provider: aws
        region: ${region-name}
        bucket: ${bucket-name}
        prefix: ${backup-path2}
  - ... # other clusters
  template:
    br:
      sendCredToTikv: false
      options:
      - --volume-type=gp3
      - --volume-iops=7000
      - --volume-throughput=400
    toolImage: ${br-image}
    warmup: sync
    warmupImage: ${wamrup-image}
```

If you grant permissions by associating ServiceAccount with IAM, you can create the `VolumeRestore` CR as follows:

```
kubectl apply -f restore-fed.yaml
```

The `restore-fed.yaml` file has the following content:

```
---
apiVersion: federation.pingcap.com/v1alpha1
kind: VolumeRestore
metadata:
  name: ${restore-name}
spec:
  clusters:
  - k8sClusterName: ${k8s-name1}
    tcName: ${tc-name1}
    tcNamespace: ${tc-namespace1}
    backup:
      s3:
        provider: aws
```

```
      region: ${region-name}
      bucket: ${bucket-name}
      prefix: ${backup-path1}
  - k8sClusterName: ${k8s-name2}
    tcName: ${tc-name2}
    tcNamespace: ${tc-namespace2}
    backup:
      s3:
        provider: aws
        region: ${region-name}
        bucket: ${bucket-name}
        prefix: ${backup-path2}
  - ... # other clusters
  template:
    br:
      sendCredToTikv: false
      options:
      - --volume-type=gp3
      - --volume-iops=7000
      - --volume-throughput=400
    toolImage: ${br-image}
    serviceAccount: tidb-backup-manager
    warmup: sync
    warmupImage: ${warmup-image}
```

Step 3. View the restore status

After creating the `VolumeRestore` CR, the restore process automatically start.

To check the restore status, use the following command:

```
kubectl get vrt -n ${namespace} -o wide
```

7.4.8.5 FAQs on EBS Snapshot Backup and Restore across Multiple Kubernetes

This document addresses common questions and solutions related to EBS snapshot backup and restore across multiple Kubernetes environments.

7.4.8.5.1 New tags on snapshots and restored volumes

Symptom: Some tags are automatically added to generated snapshots and restored EBS volumes

Explanation: The new tags are added for traceability. Snapshots inherit all tags from the individual source EBS volumes, while restored EBS volumes inherit tags from the source

snapshots but prefix keys with `snapshot\`. Additionally, new tags such as `<TiDBCluster-BR: true>`, `<snapshot/createdFromSnapshotId, {source-snapshot-id}>` are added to restored EBS volumes.

7.4.8.5.2 Backup Initialize Failed

Symptom: You get the error that contains `GC safepoint 443455494791364608 exceed TS 0` when the backup is initializing.

Solution: This issue might occur if you have disabled the feature of "resolved ts" in TiKV or PD. Check the configuration of TiKV and PD:

- For TiKV, confirm if you set `resolved-ts.enable = false` or `raftstore.report-min-resolved-ts-interval = "0s"`. If so, remove these configurations.
- For PD, confirm if you set `pd-server.min-resolved-ts-persistence-interval = "0s"`. If so, remove this configuration.

7.4.8.5.3 Backup failed due to execution twice

Issue: #5143

Symptom: You get the error that contains `backup meta file exists`, and the backup pod is scheduled twice.

Solution: This issue might occur if the first backup pod is evicted by Kubernetes due to node resource pressure. You can configure `PriorityClass` and `ResourceRequirements` to reduce the possibility of eviction. For more details, refer to the comment of issue.

7.4.8.5.4 Save time for backup by controlling snapshot size calculation level

Symptom: Scheduled backup can't be completed in the expected window due to the cost of snapshot size calculation.

Solution: By default, both full size and incremental size are calculated by calling the AWS service, which might take several minutes. You can set `spec.template.calcSizeLevel` to `full` to skip incremental size calculation, set it to `incremental` to skip full size calculation, and set it to `none` to skip both calculations.

7.5 Maintain

7.5.1 Restart a TiDB Cluster on 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 Pods in a TiDB component and how to perform a graceful restart to a single TiKV Pod.

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.

7.5.1.1 Perform 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: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    ...
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
  tikv:
    ...
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
  tidb:
    ...
    annotations:
      tidb.pingcap.com/restartedAt: 2020-04-20T12:00
```

7.5.1.2 Perform a graceful restart to a single TiKV Pod

Starting from v1.2.5, TiDB Operator supports graceful restart for a single TiKV Pod.

To trigger a graceful restart, add an annotation with the `tidb.pingcap.com/evict-leader` key:

```
kubectl -n ${namespace} annotate pod ${tikv_pod_name} tidb.pingcap.com/evict-leader="delete-pod"
```

When the number of TiKV region leaders drops to zero, according to the value of this annotation, TiDB Operator might have different behaviors:

- `none`: TiDB Operator does nothing.
- `delete-pod`: TiDB Operator deletes the Pod by taking the following steps:
    1. TiDB Operator calls the PD API and adds evict-leader-scheduler for the TiKV store.
    2. When the number of TiKV region leaders drops to zero, TiDB Operator deletes the Pod and recreates it.
    3. When the new Pod becomes ready, remove the evict-leader-scheduler for the TiKV store by calling the PD API.

7.5.2 Destroy TiDB Clusters on Kubernetes

This document describes how to destroy TiDB clusters on Kubernetes.

7.5.2.1 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}
```

7.5.2.2 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}
```

7.5.2.3 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"}}'
```

7.5.3 View TiDB Logs on Kubernetes

This document introduces the methods to view logs of TiDB components and TiDB slow log.

7.5.3.1 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`.

7.5.3.2 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.

7.5.4 Modify TiDB Cluster Configuration

For a TiDB cluster, you can update the configuration of components online using SQL statements, including TiDB, TiKV, and PD, without restarting the cluster components. However, for TiDB clusters deployed on Kubernetes, after you upgrade or restart the cluster, the configurations updated using SQL statements will be overwritten by those in the `TidbCluster` CR. This leads to the online configuration update being invalid.

This document describes how to modify the configuration of TiDB clusters deployed on Kubernetes. Due to the special nature of PD, you need to separately modify the configuration of PD and other components.

7.5.4.1 Modify configuration for TiDB, TiKV, and other components

For TiDB and TiKV, if you modify their configuration online using SQL statements, after you upgrade or restart the cluster, the configurations will be overwritten by those in the `TidbCluster` CR. This leads to the online configuration update being invalid. Therefore, to persist the configuration, you must directly modify their configurations in the `TidbCluster` CR.

For TiFlash, TiProxy, TiCDC, and Pump, you can only modify their configurations in the `TidbCluster` CR.

To modify the configuration in the `TidbCluster` CR, take the following steps:

1. Refer to the parameters in Configure TiDB components to modify the component configuration in the `TidbCluster` CR:

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

2. After the configuration is modified, view the updating progress:

   ```
   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.

7.5.4.2 Modify PD configuration

After PD is started for the first time, some PD configuration items are persisted in etcd. The persisted configuration in etcd takes precedence over the configuration file in PD. Therefore, after the first start, you cannot modify some PD configuration by using the `TidbCluster` CR.

Among all the PD configuration items listed in Modify PD configuration online, after the first start, only `log.level` can be modified by using the `TidbCluster` CR. Other configurations cannot be modified by using CR.

For TiDB clusters deployed on Kubernetes, if you need to modify the PD configuration, you can modify the configuration online using SQL statements, pd-ctl, or PD server API.

7.5.4.2.1 Modify PD microservice configuration

> **Note:**
>
> Starting from v8.0.0, PD supports the microservice mode (experimental).

After each component of the PD microservices is started for the first time, some PD configuration items are persisted in etcd. The persisted configuration in etcd takes precedence over the configuration file in PD. Therefore, after the first start of each PD microservice component, you cannot modify some PD configuration items by using the `TidbCluster` CR.

Among all the configuration items of PD microservices listed in Modify PD configuration dynamically, after the first start of each PD microservice component, only `log.level` can be modified by using the `TidbCluster` CR. Other configurations cannot be modified by using CR.

For TiDB clusters deployed on Kubernetes, if you need to modify configuration items of PD microservices, you can modify them dynamically using SQL statements, pd-ctl, or PD server API.

7.5.4.3 Modify TiProxy configuration

Modifying the configuration of the TiProxy component never restarts the Pod. If you want to restart the Pod, you need to manually kill the Pod or change the Pod image to manually trigger the restart.

7.5.5 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.

7.5.5.1 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.

7.5.5.2 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.

7.5.5.2.1 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.

7.5.5.2.2 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.

7.5.5.2.3 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 refer to the following two methods:

- Method 1: Configure `spec.tikv.recoverFailover: true` (Supported since TiDB Operator v1.1.5).

```
kubectl patch tc -n ${namespace} ${cluster_name} --type merge -p '{"
    ↪ spec":{"tikv":{"recoverFailover": true}}}'
```

  Every time after the cluster recovers from failover, TiDB Operator automatically scales in the newly created Pods.

- Method 2: Configure `spec.tikv.failover.recoverByUID: ${recover_uid}`.

  `${recover_uid}` is the UID of this failover. You can get the UID by running the following command:

```
kubectl get tc -n ${namespace} ${cluster_name} -ojsonpath='{.status.
    ↪ tikv.failoverUID}'
```

  TiDB Operator automatically scales in the newly created TiKV Pods according to `${recover_uid}`.

7.5.5.2.4 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 refer to the following two methods:

- Method 1: Configure `spec.tiflash.recoverFailover: true` (Supported since TiDB Operator v1.1.5).

```
kubectl patch tc -n ${namespace} ${cluster_name} --type merge -p '{"
    ↪ spec":{"tiflash":{"recoverFailover": true}}}'
```

  Every time after the cluster recovers from failover, TiDB Operator automatically scales in the newly created Pods.

- Method 2: Configure `spec.tiflash.failover.recoverByUID: ${recover_uid}`.

  `${recover_uid}` is the UID of this failover. You can get the UID by running the following command:

```
kubectl get tc -n ${namespace} ${cluster_name} -ojsonpath='{.status.
    ↪ tiflash.failoverUID}'
```

  TiDB Operator automatically scales in the newly created TiFlash Pods according to `${recover_uid}`.

7.5.5.2.5 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.

7.5.6 Pause Sync of a TiDB Cluster on Kubernetes

This document introduces how to pause sync of a TiDB cluster on Kubernetes using configuration.

7.5.6.1 What is sync in TiDB Operator

In TiDB Operator, controller regulates the state of the TiDB cluster on 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.

7.5.6.2 Use scenarios

Here are some cases where you might need to pause sync of a TiDB cluster on 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.

7.5.6.3 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 patch tc ${cluster_name} -n ${namespace} --type merge -p '{"
    ↪ spec":{"paused": true}}'
```

2. 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
```

7.5.6.4 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 patch tc ${cluster_name} -n ${namespace} --type merge -p '{"
   ↪ spec":{"paused": false}}'
```

2. 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)
```

7.5.7 Suspend TiDB cluster

This document introduces how to suspend the TiDB cluster or suspend the TiDB cluster components on Kubernetes by configuring the `TidbCluster` object. After suspending the cluster, you can stop the Pods of all components or one specific component and retain the `TidbCluster` object and other resources (such as Service and PVC).

In some test scenarios, if you need to save resources, you can suspend the TiDB cluster when you are not using it.

> **Note:**
>
> To suspend the TiDB cluster, the TiDB Operator version must be >= v1.3.7.

7.5.7.1 Configure TiDB cluster suspending

If you need to suspend the TiDB cluster, take the following steps:

1. In the `TidbCluster` object, configure `spec.suspendAction` field to suspend the entire TiDB cluster:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  suspendAction:
    suspendStatefulSet: true
  # ...
```

   TiDB Operator also supports suspending one or more components in TiDB clusters. Taking TiKV as an example, you can suspend TiKV in the TiDB cluster by configuring `spec.tikv.suspendAction` field in the `TidbCluster` object:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  tikv:
    suspendAction:
      suspendStatefulSet: true
  # ...
```

2. After suspending the TiDB cluster, you can run the following command to observe that the Pods of the suspended component are gradually deleted.

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

Pods of each suspended component will be deleted in the following order:

- TiDB
- TiFlash
- TiCDC
- TiKV
- Pump
- TiProxy
- PD

Note:

If PD microservices (introduced in TiDB v8.0.0) are deployed in a cluster, the Pods of PD microservices are deleted after the PD Pods are deleted.

7.5.7.2 Restore TiDB cluster

After a TiDB cluster or its component is suspended, if you need to restore the TiDB cluster, take the following steps:

1. In the `TidbCluster` object, configure the `spec.suspendAction` field to restore the entire suspended TiDB cluster:

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
```

```
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  suspendAction:
    suspendStatefulSet: false
  # ...
```

TiDB Operator also supports restoring one or more components in the TiDB cluster. Taking TiKV as an example, you can restore TiKV in the TiDB cluster by configuring `spec.tikv.suspendAction` field in the `TidbCluster` object.

```
apiVersion: pingcap.com/v1alpha1
kind: TidbCluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  tikv:
    suspendAction:
      suspendStatefulSet: false
  # ...
```

2. After restoring the TiDB cluster, you can run the following command to observe that the Pods of the suspended component are gradually created.

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

7.5.8 Maintain Different TiDB Clusters Separately Using Multiple Sets of TiDB Operator

You can use one set of TiDB Operator to manage multiple TiDB clusters. If you have the following application needs, you can deploy multiple sets of TiDB Operator to manage different TiDB clusters:

- You need to perform a canary upgrade on TiDB Operator so that the potential issues of the new version do not affect your application.
- Multiple TiDB clusters exist in your organization, and each cluster belongs to different teams. Each team needs to manage their own cluster.

This document describes how to deploy multiple sets of TiDB Operator to manage different TiDB clusters.

When you use TiDB Operator, `tidb-scheduler` is not mandatory. Refer to tidb-scheduler and default-scheduler to confirm whether you need to deploy `tidb-scheduler`.

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.

7.5.8.1 Deploy multiple sets of TiDB Operator

1. Deploy the first set of TiDB Operator.

   Refer to Deploy TiDB Operator - Customize TiDB Operator to deploy the first set of TiDB Operator. Add the following configuration in the `values.yaml`:

   ```
   controllerManager:
     selector:
     - user=dev
   ```

2. Deploy the first TiDB cluster.

   1. Refer to Configure the TiDB Cluster - Configure TiDB deployment 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 on General Kubernetes to deploy the TiDB cluster. Confirm that each component in the cluster is started normally.

3. Deploy the second set of TiDB Operator.

   Refer to Deploy TiDB Operator to deploy the second set of 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:
  # If you do not need tidb-scheduler, set this value to false.
  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 second 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 on 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.

If you want to deploy a third or more sets of TiDB Operator, repeat step 3, step 4, and step 5.

7.5.8.2 Related parameters

In the `values.yaml` file in the `tidb-operator` chart, the following parameters are related to the deployment of multiple sets of TiDB Operator:

- `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`.

7.5.9 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.

7.5.9.1 Prerequisites

- kubectl
- 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.

7.5.9.2 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.

7.5.9.3 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}
```

7.5.9.4 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.

7.5.9.4.1 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
```

7.5.9.4.2 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
```

7.5.9.5 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.

7.5.9.5.1 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. Add annotation with a `tidb.pingcap.com/evict-leader` key to the TiKV Pod to trigger the graceful restart. After TiDB Operator evicts the TiKV Region Leader, TiDB Operator deletes the Pod.

```
kubectl -n ${namespace} annotate pod ${pod_name} tidb.pingcap.com/evict
    ↪ -leader="delete-pod"
```

4. Confirm that the TiKV Pod is successfully scheduled to another node:

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

5. Confirm that the Region Leader is transferred back:

```
kubectl -n ${namespace} get tc ${cluster_name} -ojson | jq ".status.
    ↪ tikv.stores | .[] | select ( .podName == \"${pod_name}\" ) | .
    ↪ leaderCount"
```

7.5.9.5.2 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. Recreate a TiKV Pod.

7.5.9.6 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.

7.5.9.7 Evict TiKV Region Leader

1. Add an annotation with a `tidb.pingcap.com/evict-leader` key to the TiKV Pod:

```
kubectl -n ${namespace} annotate pod ${pod_name} tidb.pingcap.com/evict
    ↪ -leader="none"
```

2. Confirm that all Region Leaders are migrated:

```
kubectl -n ${namespace} get tc ${cluster_name} -ojson | jq ".status.
    ↪ tikv.stores | .[] | select ( .podName == \"${pod_name}\" ) | .
    ↪ leaderCount"
```

   If the output is `0`, all Region Leaders are successfully migrated.

7.5.9.8 Recreate a TiKV Pod

1. Evict the TiKV Region Leader to another Pod.
2. 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 -n ${namespace} tc ${cluster_name} -ojson | jq ".status
    ↪ .tikv.stores | .[] | select ( .podName == \"${pod_name}\" )
    ↪ | .id"
```

2. In any of the PD Pods, use `pd-ctl` command to take the TiKV Pod offline:

```
kubectl exec -n ${namespace} ${cluster_name}-pd-0 -- /pd-ctl store
    ↪ delete ${store_id}
```

3. Wait for the store status (`state_name`) to become `Tombstone`:

```
kubectl exec -n ${namespace} ${cluster_name}-pd-0 -- watch /pd-ctl
    ↪ store ${store_id}
```

   Expected output

```
{
  "store": {
    "id": "${store_id}",
    // ...
    "state_name": "Tombstone"
  },
  // ...
}
```

3. Unbind the TiKV Pod with the currently used storage.

    1. Check the `PersistentVolumeClaim` used by the Pod:

    ```
    kubectl -n ${namespace} get pvc -l tidb.pingcap.com/pod-name=${
        ↪ pod_name}
    ```

    Expected output

    The NAME field is the name of PVC.

    ```
    NAME         STATUS  VOLUME                                     CAPACITY
        ↪  ACCESS MODES     STORAGECLASS AGE
    ${pvc_name} Bound   pvc-a8f16ca6-a675-448f-82c3-3cae624aa0e2 100Gi
        ↪   RWO              standard     18m
    ```

    2. Delete the `PersistentVolumeClaim`:

    ```
    kubectl delete -n ${namespace} pvc ${pvc_name} --wait=false
    ```

4. Delete the old TiKV Pod and wait for the new TiKV Pod to join the cluster.

```
kubectl delete -n ${namespace} pod ${pod_name}
```

Wait for the state of the new TiKV Pod to become `Up`.

```
kubectl get -n ${namespace} tc ${cluster_name} -ojson | jq ".status.
    ↪ tikv.stores | .[] | select ( .podName == \"${pod_name}\" )"
```

Expected output

```
{
  "id": "${new_store_id}",
  "ip": "${pod_name}.${cluster_name}-tikv-peer.default.svc",
  "lastTransitionTime": "2022-03-08T06:39:58Z",
  "leaderCount": 3,
  "podName": "${pod_name}",
  "state": "Up"
}
```

As you can see from the output, the new TiKV Pod have a new `store-id`, and Region Leaders are migrated to this TiKV Pod automatically.

5. Remove the useless evict-leader-scheduler:

```
kubectl exec -n ${namespace} ${cluster_name}-pd-0 -- /pd-ctl scheduler
    ↪ remove evict-leader-scheduler-${store_id}
```

7.5.10 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.

7.5.10.1 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 repositories 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 repository 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.

7.5.11 Replace Nodes for a TiDB Cluster

7.5.11.1 Replace Nodes for a TiDB Cluster on Cloud Disks

This document describes a method for replacing and upgrading nodes without downtime for a TiDB cluster that uses cloud storage. You can change the nodes to a higher configuration, or upgrade the nodes to a newer version of Kubernetes.

This document uses Amazon EKS as an example and describes how to create a new node group and migrate a TiDB cluster to the new node group using a rolling restart. You can

use this method to replace a node group with more compute resources for TiKV or TiDB and upgrade EKS.

For other cloud platforms, refer to Google Cloud GKE or Azure AKS and operate on the node group.

7.5.11.1.1 Prerequisites

- A TiDB cluster is deployed on the cloud. If not, refer to Deploy on Amazon EKS and deploy a cluster.
- The TiDB cluster uses cloud storage as its data disk.

7.5.11.1.2 Step 1: Create new node groups

1. Locate the `cluster.yaml` configuration file for the EKS cluster that the TiDB cluster is deployed in, and save a copy of the file as `cluster-new.yaml`.
2. In `cluster-new.yaml`, add new groups (for example, `tidb-1b-new` and `tikv-1a-new`):

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: your-eks-cluster
  region: ap-northeast-1

nodeGroups:
...
  - name: tidb-1b-new
    desiredCapacity: 1
    privateNetworking: true
    availabilityZones: ["ap-northeast-1b"]
    instanceType: c5.4xlarge
    labels:
      dedicated: tidb
    taints:
      dedicated: tidb:NoSchedule
  - name: tikv-1a-new
    desiredCapacity: 1
    privateNetworking: true
    availabilityZones: ["ap-northeast-1a"]
    instanceType: r5b.4xlarge
    labels:
      dedicated: tikv
    taints:
      dedicated: tikv:NoSchedule
```

Note:

- `availabilityZones` must be the same as that of the original node group to be replaced.
- The `tidb-1b-new` and `tikv-1a-new` node groups configured in the YAML above are only for demonstration. You need to configure the node groups according to your needs.

If you want to scale up a node, modify `instanceType`. If you want to upgrade the Kubernetes version, first upgrade the version of your cluster control plane. For details, see Updating a Cluster.

3. In `cluster-new.yaml`, delete the original node groups to be replaced.

   In this example, delete `tidb-1b` and `tikv-1a`. You need to delete node groups according to your needs.

4. In `cluster.yaml`, delete the node groups that **are not to be replaced** and keep the node groups that are to be replaced. The retained node groups will be deleted from the cluster.

   In this example, keep `tidb-1a` and `tikv-1b`, and delete other node groups. You need to keep or delete node groups according to your needs.

5. Create the new node groups:

```
eksctl create nodegroup -f cluster_new.yml
```

Note:

This command only creates new node groups. Node groups that already exist are ignored and not created again. The command does not delete the node groups that do not exist.

6. Confirm that the new nodes are added to the cluster.

```
kubectl get no -l alpha.eksctl.io/nodegroup-name=${new_nodegroup1}
kubectl get no -l alpha.eksctl.io/nodegroup-name=${new_nodegroup2}
...
```

`${new_nodegroup}` is the name of a new node group. In this example, the new node groups are `tidb-1b-new` and `tikv-1a-new`. You need to configure the node group name according to your needs.

7.5.11.1.3 Step 2: Mark the original nodes as non-schedulable

You need to mark the original nodes as non-schedulable to ensure that no new Pod is scheduled to it. Run the `kubectl cordon` command:

```
kubectl cordon -l alpha.eksctl.io/nodegroup-name=${origin_nodegroup1}
kubectl cordon -l alpha.eksctl.io/nodegroup-name=${origin_nodegroup2}
...
```

`${origin_nodegroup}` is the name of an original node group. In this example, the original node groups are `tidb-1b` and `tikv-1a`. You need to configure the node group name according to your needs.

7.5.11.1.4 Step 3: Rolling restart the TiDB cluster

Refer to Restart a TiDB Cluster on Kubernetes and perform a rolling restart on the TiDB cluster.

7.5.11.1.5 Step 4: Delete the original node groups

Check whether there are TiDB, PD, or TiKV Pods left on nodes of the original node groups:

```
kubectl get po -n ${namespace} -owide
```

If no TiDB, PD, or TiKV Pods are left on the nodes of the original node groups, you can delete the original node groups:

```
eksctl delete nodegroup -f cluster.yaml --approve
```

7.5.11.2 Replace Nodes for a TiDB Cluster on Local Disks

This document describes a method for replacing and upgrading nodes without downtime for a TiDB cluster that uses local storage.

> **Note:**
>
> If you only need to maintain a specific node in the TiDB cluster, refer to Maintain Kubernetes Nodes that Hold the TiDB Cluster.

7.5.11.2.1 Prerequisites

- A TiDB cluster is deployed. If not, refer to Deploy TiDB on General Kubernetes and deploy a cluster.
- The new node is ready and joins the Kubernetes cluster.

7.5.11.2.2 Step 1: Clone the configuration of the original TiDB cluster

1. Export a copy of the cluster configuration file, `tidb-cluster-clone.yaml`, by running the following command:

```
kubectl get tidbcluster ${origin_cluster_name} -n ${namespace} -oyaml >
    ↪ tidb-cluster-clone.yaml
```

`${origin_cluster_name}` is the name of the original cluster. `${namespace}` is the namespace of the original cluster.

2. Modify `tidb-cluster-clone.yaml` and allow the new clone cluster to join the original TiDB cluster.

```
kind: TidbCluster
metadata:
  name: ${clone_cluster_name}
spec:
  cluster:
    name: ${origin_cluster_name}
...
```

`${clone_cluster_name}` is the name of the clone cluster. `${origin_cluster_name}` is the name of the original cluster.

7.5.11.2.3 Step 2: Sign certificates for the clone cluster

If the original cluster enables TLS, you need to sign certificates for the clone cluster. If not, you can skip this step and move to Step 3.

Use cfssl

If you use cfssl to sign certificates, you must sign certificates using the same certification authority (CA) as the original cluster. To complete the signing process, follow instructions in step 5~7 in Using cfssl.

Use cert-manager

If you use cert-manager, you must sign certificates using the same Issuer as the original cluster. To complete the signing process, follow instructions in step 3 in Using cert-manager.

7.5.11.2.4 Step 3: Mark the nodes to be replaced as non-schedulable

By marking the nodes to be replaced as non-schedulable, you can ensure that no new Pod is scheduled to the nodes. Run the `kubectl cordon` command:

```
kubectl cordon ${replace_nodename1} ${replace_nodename2} ...
```

7.5.11.2.5 Step 4: Create the clone cluster

1. Create the clone cluster by running the following command:

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

2. Confirm that the new TiDB cluster that consists of the clone cluster and the original cluster is running normally.

    - Obtain the count and state of stores in the new cluster:

```
# store count
pd-ctl -u http://<address>:<port> store | jq '.count'
# store state
pd-ctl -u http://<address>:<port> store | jq '.stores | .[] | .
    ↪ store.state_name'
```

    - Access the TiDB cluster via MySQL client.

7.5.11.2.6 Step 5: Scale in all TiDB nodes of the original cluster

Scale in all TiDB nodes of the original cluster to 0. For details, refer to Horizontal scaling.

> **Note:**
>
> If you access the original TiDB cluster via a load balancer or database middleware, before scaling in the original TiDB cluster, you need to modify the configuration to route your application traffic to the target TiDB cluster. Otherwise, your application might be affected.

7.5.11.2.7 Step 6: Scale in all TiKV nodes of the original cluster

Scale in all TiKV nodes of the original cluster to 0. For details, refer to Horizontal scaling.

7.5.11.2.8 Step 7: Scale in all PD nodes of the original cluster

Scale in all PD nodes of the original cluster to 0. For details, refer to Horizontal scaling.

7.5.11.2.9 Step 8: Delete the `spec.cluster` field in the clone cluster

Delete the `spec.cluster` field in the clone cluster by running the following command:

```
kubectl patch -n ${namespace} tc ${clone_cluster_name} --type=json -p '[{"
    ↪ op":"remove", "path":"/spec/cluster"}]'
```

`${namespace}` is the name of the clone cluster (unchanged). `${clone_cluster_name}` is the name of the clone cluster.

7.5.11.2.10 Step 9: Delete the cluster, data, and nodes of the original cluster

1. Delete the `TidbCluster` of the original cluster:

   ```
   kubectl delete -n ${namespace} tc ${origin_cluster_name}
   ```

   `${namespace}` is the name of the original cluster (unchanged). `${original_cluster_name ↪ }` is the name of the original cluster.

2. Delete the data of the original cluster. For details, refer to Delete PV and data.

3. Delete the nodes to be replaced from the Kubernetes cluster:

   ```
   kubectl delete node ${replace_nodename1} ${replace_nodename2} ...
   ```

7.6 Disaster Recovery

7.6.1 Recover the Deleted Cluster

This document describes how to recover a TiDB cluster that has been deleted mistakenly on Kubernetes. If you have mistakenly deleted a TiDB cluster using TidbCluster, you can use the method introduced in this document to recover the cluster.

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
```

7.6.2 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. For detailed introduction of this tool, see TiDB documentation - PD Recover. This document introduces how to download PD Recover and how to use it to recover a PD cluster.

7.6.2.1 Download PD Recover

1. Download the official TiDB package:

```
wget https://download.pingcap.org/tidb-community-toolkit-${version}-
    ↪ linux-amd64.tar.gz
```

   In the command above, `${version}` is the version of the TiDB cluster, such as `v8.5.0`.

2. Unpack the TiDB package:

```
tar -xzf tidb-community-toolkit-${version}-linux-amd64.tar.gz
tar -xzf tidb-community-toolkit-${version}-linux-amd64/pd-recover-${
    ↪ version}-linux-amd64.tar.gz
```

   `pd-recover` is in the current directory.

7.6.2.2 Scenario 1: At least one PD node is alive

This section introduces how to recover the PD cluster using PD Recover and alive PD nodes. This section is only applicable to the scenario where the PD cluster has alive PD nodes. If all PD nodes are unavailable, refer to Scenario 2.

> **Note:**
>
> If you restore the cluster by using alive PD, the cluster can keep all the configuration information that has taken effect in PD before the restoration.

7.6.2.2.1 Step 1. Recover the PD Pod

> **Note:**
>
> This document takes pd-0 as an example. If you use other PD pods, modify the corresponding command.

Use an alive PD node `pd-0` to force recreate the PD cluster. The detailed steps are as follows:

1. Let pd-0 pod enter debug mode:

```
kubectl annotate pod ${cluster_name}-pd-0 -n ${namespace} runmode=debug
kubectl exec ${cluster_name}-pd-0 -n ${namespace} -- kill -SIGTERM 1
```

2. Enter the pd-0 pod:

```
kubectl exec ${cluster_name}-pd-0 -n ${namespace} -it -- sh
```

3. Refer to the default startup script pd-start-script or the start script of an alive PD node, and configure environment variables in pd-0:

```
# Use HOSTNAME if POD_NAME is unset for backward compatibility.
POD_NAME=${POD_NAME:-$HOSTNAME}
# the general form of variable PEER_SERVICE_NAME is: "<clusterName>-pd-
    ↪ peer"
cluster_name=`echo ${PEER_SERVICE_NAME} | sed 's/-pd-peer//'`
domain="${POD_NAME}.${PEER_SERVICE_NAME}.${NAMESPACE}.svc"
discovery_url="${cluster_name}-discovery.${NAMESPACE}.svc:10261"
encoded_domain_url=`echo ${domain}:2380 | base64 | tr "\n" " " | sed "s
    ↪ / //g"`
elapseTime=0
period=1
threshold=30
while true; do
sleep ${period}
elapseTime=$(( elapseTime+period ))

if [[ ${elapseTime} -ge ${threshold} ]]
then
echo "waiting for pd cluster ready timeout" >&2
exit 1
fi

if nslookup ${domain} 2>/dev/null
then
echo "nslookup domain ${domain}.svc success"
break
else
echo "nslookup domain ${domain} failed" >&2
fi
done
```

```
ARGS="--data-dir=/var/lib/pd \
--name=${POD_NAME} \
--peer-urls=http://0.0.0.0:2380 \
--advertise-peer-urls=http://${domain}:2380 \
--client-urls=http://0.0.0.0:2379 \
--advertise-client-urls=http://${domain}:2379 \
--config=/etc/pd/pd.toml \
"

if [[ -f /var/lib/pd/join ]]
then
# The content of the join file is:
#   demo-pd-0=http://demo-pd-0.demo-pd-peer.demo.svc:2380,demo-pd-1=
    ↪ http://demo-pd-1.demo-pd-peer.demo.svc:2380
# The --join args must be:
#   --join=http://demo-pd-0.demo-pd-peer.demo.svc:2380,http://demo-pd
    ↪ -1.demo-pd-peer.demo.svc:2380
join=`cat /var/lib/pd/join | tr "," "\n" | awk -F'=' '{print $2}' | tr
    ↪ "\n" ","`
join=${join%,}
ARGS="${ARGS} --join=${join}"
elif [[ ! -d /var/lib/pd/member/wal ]]
then
until result=$(wget -qO- -T 3 http://${discovery_url}/new/${
    ↪ encoded_domain_url} 2>/dev/null); do
echo "waiting for discovery service to return start args ..."
sleep $((RANDOM % 5))
done
ARGS="${ARGS}${result}"
fi
```

4. Use original pd-0 data directory to force start a new PD cluster:

```
echo "starting pd-server ..."
sleep $((RANDOM % 10))
echo "/pd-server --force-new-cluster ${ARGS}"
exec /pd-server --force-new-cluster ${ARGS} &
```

5. Exit pd-0 pod:

```
exit
```

6. Execute the following command to confirm that PD is started:

```
kubectl logs -f ${cluster_name}-pd-0 -n ${namespace} | grep "Welcome to
    ↪  Placement Driver (PD)"
```

7.6.2.2.2 Step 2. Recover the PD cluster

1. Copy `pd-recover` to the PD pod:

```
kubectl cp ./pd-recover ${namespace}/${cluster_name}-pd-0:./
```

2. Recover the PD cluster by running the `pd-recover` command:

   In the command, use the newly created cluster in the previous step:

```
kubectl exec ${cluster_name}-pd-0 -n ${namespace} -- ./pd-recover --
    ↪ from-old-member -endpoints http://127.0.0.1:2379
```

```
recover success! please restart the PD cluster
```

7.6.2.2.3 Step 3. Restart the PD Pod

1. Delete the PD Pod:

```
kubectl delete pod ${cluster_name}-pd-0 -n ${namespace}
```

2. Confirm the Cluster ID is generated:

```
kubectl -n ${namespace} exec -it ${cluster_name}-pd-0 -- wget -q http
    ↪ ://127.0.0.1:2379/pd/api/v1/cluster
kubectl -n ${namespace} exec -it ${cluster_name}-pd-0 -- cat cluster
```

7.6.2.2.4 Step 4. Recreate other failed or available PD nodes

In this example, recreate pd-1 and pd-2:

```
kubectl -n ${namespace} delete pvc pd-${cluster_name}-pd-1 --wait=false
kubectl -n ${namespace} delete pvc pd-${cluster_name}-pd-2 --wait=false

kubectl -n ${namespace} delete pod ${cluster_name}-pd-1
kubectl -n ${namespace} delete pod ${cluster_name}-pd-2
```

7.6.2.2.5 Step 5. Check PD health and configuration

Check health:

```
kubectl -n ${namespace} exec -it ${cluster_name}-pd-0 -- ./pd-ctl health
```

Check configuration. The following command uses placement rules as an example:

```
kubectl -n ${namespace} exec -it ${cluster_name}-pd-0 -- ./pd-ctl config
    ↪ placement-rules show
```

Now the TiDB cluster is recovered.

7.6.2.3 Scenarios 2: All PD nodes are down and cannot be recovered

This section introduces how to recover the PD cluster by using PD Recover and creating new PD nodes. This section is only applicable when all PD nodes in the cluster have failed and cannot be recovered. If there are alive PD nodes in the cluster, refer to Scenario 1.

> **Warning:**
>
> If you restore the cluster by creating new PD nodes, the cluster will lose all the configuration information that has taken effect in PD before the restoration.

7.6.2.3.1 Step 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
```

7.6.2.3.2 Step 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`.

7.6.2.3.3 Step 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 patch tc ${cluster_name} -n ${namespace} --type merge -p '{"
       ↪ spec":{"pd":{"replicas": 0}}}'
   ```

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 patch sts ${cluster_name}-pd -n ${namespace} -p '{"spec":{"
    ↪ replicas": 0}}'
```

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 patch tc ${cluster_name} -n ${namespace} --type merge -p '{"
    ↪ spec":{"pd":{"replicas": 1}}}'
```

   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 patch sts ${cluster_name}-pd -n ${namespace} -p '{"spec":{"
    ↪ replicas": 1}}'
```

   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)"
```

7.6.2.3.4 Step 4. Recover the cluster

1. Copy `pd-recover` command to the PD pod:

```
kubectl cp ./pd-recover ${namespace}/${cluster_name}-pd-0:./
```

2. Execute the `pd-recover` command to recover the PD cluster:

```
kubectl exec ${cluster_name}-pd-0 -n ${namespace} -- ./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
```

7.6.2.3.5 Step 5. Restart the PD Pod

1. Delete the PD Pod:

```
kubectl delete pod ${cluster_name}-pd-0 -n ${namespace}
```

2. Execute the following command to confirm the Cluster ID is the one got in Get Cluster ID.

```
kubectl -n ${namespace} exec -it ${cluster_name}-pd-0 -- wget -q http
    ↪ ://127.0.0.1:2379/pd/api/v1/cluster
kubectl -n ${namespace} exec -it ${cluster_name}-pd-0 -- cat cluster
```

7.6.2.3.6 Step 6. Scale out the PD cluster

Execute the following command to set the value of `spec.pd.replicas` to the desired number of Pods:

```
kubectl patch tc ${cluster_name} -n ${namespace} --type merge -p '{"spec":{"
    ↪ pd":{"replicas": $replicas}}}'
```

7.6.2.3.7 Step 7. Restart TiDB and TiKV

Use the following commands to restart the TiDB and TiKV clusters:

```
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}
```

Now the TiDB cluster is recovered.

8 Troubleshoot

8.1 Tips for troubleshooting TiDB on Kubernetes

This document describes the commonly used tips for troubleshooting TiDB on Kubernetes.

8.1.1 Use the debug 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 debug 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 debug 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 debug mode.

> **Note:**
>
> If Pod is running, you can force restart the container by running the following command.
>
> ```
> kubectl exec ${pod_name} -n ${namespace} -c ${container} --
>     ↪ kill -SIGTERM 1
> ```

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.

8.1.2 Modify the configuration of a TiKV instance

In some test scenarios, if you need to modify the configuration of a TiKV instance and do not want the configuration to affect other instances, you can use the following methods.

8.1.2.1 Modify online

Refer to the document and use SQL to online modify the configuration of a single TiKV instance.

> **Note:**
>
> The modification made by this method is temporary and not persistent. After the Pod is restarted, the original configuration will be used.

8.1.2.2 Modify manually in debug mode

After the TiKV Pod enters debug mode, you can modify the TiKV configuration file and then manually start the TiKV process using the modified configuration file.

The steps are as follows:

1. Get the start command from the TiKV log, which will be used in a subsequent step.

```
kubectl logs pod ${pod_name} -n ${namespace} -c tikv | head -2 | tail
    ↪ -1
```

You can see a similar output as follows, which is the start command of TiKV.

```
/tikv-server --pd=http://${tc_name}-pd:2379 --advertise-addr=${pod_name
    ↪ }.${tc_name}-tikv-peer.default.svc:20160 --addr=0.0.0.0:20160 --
    ↪ status-addr=0.0.0.0:20180 --data-dir=/var/lib/tikv --capacity=0
    ↪ --config=/etc/tikv/tikv.toml
```

> **Note:**
>
> If the TiKV Pod is in the `CrashLoopBackoff` state, you cannot get the start command from the log. In such cases, you might splice the start command according to the above command format.

2. Turn on debug mode for the Pod and restart the Pod.

   Add an annotation to the Pod and wait for the Pod to restart.

```
kubectl annotate pod ${pod_name} -n ${namespace} runmode=debug
```

   If the Pod keeps running, you can force restart the container by running the following command:

```
kubectl exec ${pod_name} -n ${namespace} -c tikv -- kill -SIGTERM 1
```

Check the log of TiKV to ensure that the Pod is in debug mode.

```
kubectl logs ${pod_name} -n ${namespace} -c tikv
```

The output is similar to the following:

```
entering debug mode.
```

3. Enter the TiKV container by running the following command:

```
kubectl exec -it ${pod_name} -n ${namespace} -c tikv -- sh
```

4. In the TiKV container, copy the configuration file of TiKV to a new file, and modify the new file.

```
cp /etc/tikv/tikv.toml /tmp/tikv.toml && vi /tmp/tikv.tmol
```

5. In the TiKV container, modify the start command obtained in Step 1 and configure the `--config` flag as the new configuration file. Run the modified start command to start the TiKV process:

```
/tikv-server --pd=http://${tc_name}-pd:2379 --advertise-addr=${pod_name
    ↪ }.${tc_name}-tikv-peer.default.svc:20160 --addr=0.0.0.0:20160 --
    ↪ status-addr=0.0.0.0:20180 --data-dir=/var/lib/tikv --capacity=0
    ↪ --config=/tmp/tikv.toml
```

After the test is completed, if you want to recover the TiKV Pod, you can delete the TiKV Pod and wait for the Pod to be automatically started.

```
kubectl delete ${pod_name} -n ${namespace}
```

8.1.3 Configure forceful upgrade for the TiKV cluster

Normally, during TiKV rolling update, TiDB Operator evicts all Region leaders for TiKV Pods before restarting the TiKV Pods. This is meant for minimizing the impact of the rolling update on user requests.

In some test scenarios, if you do not need to wait for the Region leader to migrate during TiKV rolling upgrade, or if you want to speed up the rolling upgrade, you can configure the `spec.tikv.evictLeaderTimeout` field in the spec of TidbCluster to a small value.

```
spec:
  tikv:
    evictLeaderTimeout: 10s
```

For more information about this field, refer to Configure graceful upgrade.

> **Warning:**
>
> Configuring forceful upgrade causes some user requests to fail. It is not recommended for a production environment.

8.1.4 Configure forceful upgrade for the TiCDC cluster

> **Warning:**
>
> Configuring forceful upgrade causes replication latency to increase. It is not recommended for a production environment.

Normally, during TiCDC rolling update, TiDB Operator drains all replication workloads for TiCDC Pods before restarting the TiCDC Pods. This is meant for minimizing the impact of the rolling update on replication latency.

In some test scenarios, if you do not need to wait for the draining to complete during TiCDC rolling upgrade, or if you want to speed up the rolling upgrade, you can configure the `spec.ticdc.gracefulShutdownTimeout` field in the spec of TidbCluster to a small value.

```
spec:
  ticdc:
    gracefulShutdownTimeout: 10s
```

For more information about this field, refer to Configure graceful upgrade.

8.2 Common Deployment Failures of TiDB on Kubernetes

This document describes the common deployment failures of TiDB on Kubernetes and their solutions.

8.2.1 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}
```

8.2.2 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.
- The certificates used by TiDB or TiProxy components are not configured.

You can check the specific reason for Pending by using the `kubectl describe pod` command:

```
kubectl describe po -n ${namespace} ${pod_name}
```

8.2.2.1 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.

8.2.2.2 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}
```

8.2.2.3 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.

8.2.3 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.

8.2.4 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`.

8.2.4.1 View the log of the current container

```
kubectl -n ${namespace} logs -f ${pod_name}
```

8.2.4.2 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.

8.2.4.3 "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.

8.2.4.4 `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
```

8.2.4.5 PD Pod `nslookup domain failed`

You should see some log of PD Pod like:

```
Thu Jan 13 14:55:52 IST 2022
;; Got recursion not available from 10.43.0.10, trying next server
;; Got recursion not available from 10.43.0.10, trying next server
;; Got recursion not available from 10.43.0.10, trying next server
Server: 10.43.0.10
Address: 10.43.0.10#53

** server can't find basic-pd-0.basic-pd-peer.default.svc: NXDOMAIN

nslookup domain basic-pd-0.basic-pd-peer.default.svc failed
```

This type of failure occurs when the cluster meets both of the following two conditions:

- There are two `nameserver` in `/etc/resolv.conf`, and the second one is not IP of CoreDNS.
- The version of PD is:
  - Greater than or equal to v5.0.5.
  - Greater than or equal to v5.1.4.
  - Greater than or equal to v5.2.4.
  - All 5.3 versions.

To address this failure, add `startUpScriptVersion` to TidbCluster as:

```
...
spec:
  pd:
    startUpScriptVersion: "v1"
...
```

This failure occurs because there is something wrong with the `nslookup` in the base image (see detail in #4379). After configuring `startUpScriptVersion` to `v1`, TiDB Operator uses `dig` to check DNS instead of using `nslookup`.

8.2.4.6 Other causes

If you cannot confirm the cause from the log and `ulimit` is also a normal value, troubleshoot the issue by using the debug mode.

8.3 Common Cluster Exceptions of TiDB on Kubernetes

This document describes the common exceptions during the operation of TiDB clusters on Kubernetes and their solutions.

8.3.1 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.

8.3.2 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.

8.4 Common Network Issues of TiDB on Kubernetes

This document describes the common network issues of TiDB on Kubernetes and their solutions.

8.4.1 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:

```
kubectl -n ${namespace} exec -it ${pod_name} -- sh
```

   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 `wget` or `curl`.

     If the `wget` or `curl` 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
```

8.4.2 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.

8.5 Troubleshoot TiDB Cluster Using PingCAP Clinic

For TiDB clusters deployed on Kubernetes using TiDB Operator, you can use PingCAP Clinic Diagnostic Service (PingCAP Clinic) to remotely troubleshoot cluster problems and locally check the cluster status using the Clinic Diag client (Diag) and the Clinic Server Platform (Clinic Server).

> **Note:**
>
> This document **only** applies to clusters deployed using TiDB Operator on Kubernetes. For clusters deployed using TiUP in a self-managed environment, see PingCAP Clinic for TiUP environments.
>
> PingCAP Clinic **does not support** collecting data from clusters deployed using TiDB Ansible.

For clusters deployed using TiDB Operator, Diag is deployed as a standalone Pod. This document describes how to use the `kubectl` command to create and deploy the Diag Pod, then to collect data and perform a quick check through the API.

8.5.1 Usage scenarios

You can easily collect data from clusters and perform a quick check using the Diag of PingCAP Clinic:

- Use Diag to collect data
- Use Diag to perform a quick check on the cluster

8.5.2 Install Diag client

The following sections describe how to install Diag.

8.5.2.1 Step 1: Prepare the environment

Before deploying Diag, make sure the following items are installed on the cluster:

- Kubernetes >= v1.24
- TiDB Operator
- PersistentVolume
- RBAC
- Helm 3

8.5.2.1.1 Install Helm

To install Helm and configure the chart repository `https://charts.pingcap.org/` maintained by PingCAP, you can refer to the Use Helm document.

> **Note:**
>
> In the following sections, `${chart_version}` refers to the version of the Diag chart, for example `v1.3.1`. You can get a list of the currently supported versions by executing the `helm search repo -l diag` command.

```
helm search repo diag
NAME         CHART VERSION APP VERSION DESCRIPTION
pingcap/diag v1.3.1       v1.3.1      Clinic Diag Helm chart for Kubernetes
```

8.5.2.1.2 Check the privilege of the user

The user used for deploying Diag is expected to have the following *Role* and *Cluster Role* resources:

Role access:

```
PolicyRule:
  Resources                               Non-Resource URLs Resource Names Verbs
  ---------                               ----------------- -------------- -----
  serviceaccounts                         []                []             [get
     ↪ create delete]
  deployments.apps                        []                []             [get
     ↪ create delete]
  rolebindings.rbac.authorization.k8s.io []                 []             [get
     ↪ create delete]
  roles.rbac.authorization.k8s.io         []                []             [get
     ↪ create delete]
  secrets                                 []                []             [get
     ↪ list create delete]
  services                                []                []             [get
     ↪ list create delete]
  pods                                    []                []             [get
     ↪ list]
  tidbclusters.pingcap.com                []                []             [get
     ↪ list]
  tidbmonitors.pingcap.com                []                []             [get
     ↪ list]
```

Cluster Role access:

```
PolicyRule:
  Resources                                       Non-Resource URLs Resource Names
     ↪ Verbs
  ---------                                       ----------------- --------------
     ↪ -----
  clusterrolebindings.rbac.authorization.k8s.io [] []             [
     ↪ get create delete]
  clusterroles.rbac.authorization.k8s.io          []                []             [
     ↪ get create delete]
  pods                                            []                []             [
     ↪ get list]
  secrets                                         []                []             [
     ↪ get list]
  services                                        []                []             [
     ↪ get list]
  tidbclusters.pingcap.com                        []                []             [
```

```
    ↪ get list]
tidbmonitors.pingcap.com                    []              []              [
    ↪ get list]
```

> **Note:**
>
> If the cluster meets the criteria of least privilege deployment, you can use a smaller set of privileges. For more information, see Least privilege deployment.

Follow these steps to check the user access:

1. Check the user's *Role* and *clusterRole*:

```
kubectl describe rolebinding -n ${namespace} | grep ${user_name} -A 7
kubectl describe clusterrolebinding -n ${namespace} | grep ${user_name}
    ↪  -A 7
```

2. Check the user's access of *Role* and *Cluster Role*:

```
kubectl describe role ${role_name} -n ${namespace}
kubectl describe clusterrole ${clusterrole_name} -n ${namespace}
```

8.5.2.2 Step 2: Log in to the Clinic Server and get an access token

When Diag uploads data, the access token is used to identify the user and ensures that the data from Diag is uploaded to the organization created by the user. You need to log in to the Clinic Server to get a token.

1. Log in to the Clinic Server.

   Go to the Clinic Server for international users and select **Continue with TiDB Account** to enter the TiDB Cloud login page. If you do not have a TiDB Cloud account, you can create one on that page.

> **Note:**
>
> Clinic Server in US only uses TiDB Cloud account to log in. Users are not required to actually use TiDB Cloud service.

   Go to the Clinic Server for users in the Chinese mainland and select **Continue with AskTUG** to enter the AskTUG community login page. If you do not have an AskTUG account, you can create one on that page.

2. Create an organization.

   Create an organization on the Clinic Server. An organization is a collection of TiDB clusters. You can upload diagnostic data to the created organization.

3. Get an access token.

   To get a token, enter the organization page and click the icon in the lower-right corner of the Clusters page, and select **Get Access Token For Diag Tool**. Make sure that you have copied and saved the displayed token information.

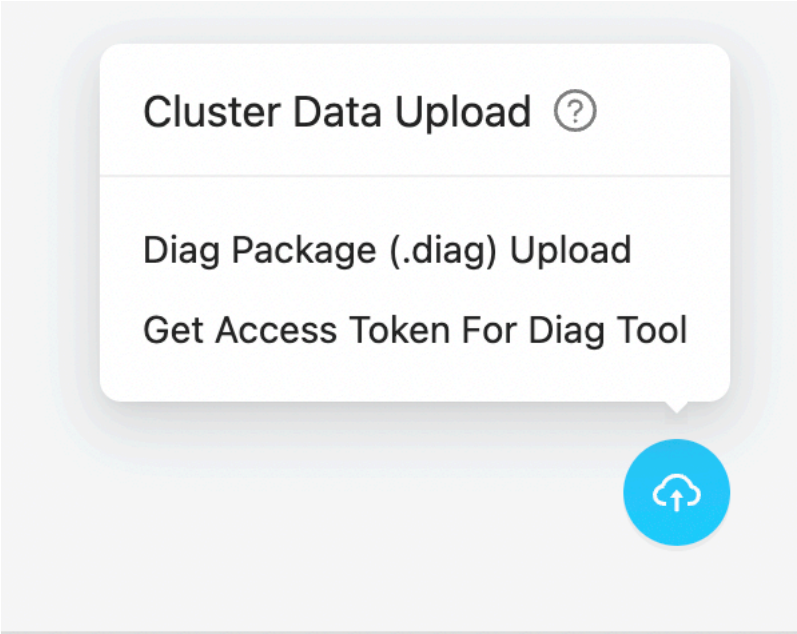


Figure 10: An example of a token

> **Note:**
> For security reasons, Clinic Server only displays the token upon the token creation. If you have lost the token, delete the old token and create a new one.

8.5.2.3 Step 3: Deploy a Diag Pod

Depending on the network connection of the cluster, you can choose one of the following methods to deploy a Diag Pod:

- Quick online deployment: If the cluster has Internet access and you would like to use the default Diag configuration, it is recommended to use the quick online deployment.
- Standard online deployment: If the cluster has Internet access and you need to customize the Diag configuration, it is recommended to use the standard online deployment.

- Offline deployment: If the cluster cannot access the Internet, you can use the offline deployment.
- Least privilege deployment: If all nodes in the cluster are running under the same namespace, you can deploy Diag to the namespace of the cluster so that Diag has the least privileges.

To use the quick online deployment, do the following:

Deploy Diag using the following `helm` command and the latest Diag image is pulled from the Docker Hub.

```
## namespace: the same as that of TiDB Operator
## diag.clinicToken: get your token in "https://clinic.pingcap.com.cn" or "
    ↪ https://clinic.pingcap.com"
helm install --namespace tidb-admin diag-collector pingcap/diag --version ${
    ↪ chart_version} \
       --set diag.clinicToken=${clinic_token}
       --set diag.clinicRegion=${clinic_region} # CN or US
```

The output is as follows:

```
NAME: diag-collector
LAST DEPLOYED: Tue Mar 15 13:00:44 2022
NAMESPACE: tidb-admin
STATUS: deployed
REVISION: 1
NOTES:
Make sure diag-collector components are running:

    kubectl get pods --namespace tidb-admin -l app.kubernetes.io/instance=
       ↪ diag-collector
    kubectl get svc --namespace tidb-admin -l app.kubernetes.io/name=diag-
       ↪ collector
```

To use the standard online deployment, do the following:

1. Get the `values-diag-collector.yaml` file from the Diag chart.

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

2. Configure the `values-diag-collector.yaml` file.

   Modify your `clinicToken` and `clinicRegion` in the `${HOME}/diag-collector/` ↪ `values-diag-collector.yaml` file.

   Other configuration parameters such as `limits`, `requests`, and `volume` can be modified according to your needs.

> **Note:**
>
> To get the token, refer to Step 2: Log in to the Clinic Server and get an access token.

3. Deploy Diag.

```
helm install diag-collector pingcap/diag --namespace=tidb-admin --
    ↪ version=${chart_version} -f ${HOME}/diag-collector/values-diag-
    ↪ collector.yaml && \
kubectl get pods --namespace tidb-admin -l app.kubernetes.io/instance=
    ↪ diag-collector
```

> **Note:**
>
> The namespace should be the same as the namespace of TiDB Operator. If TiDB Operator is not deployed, deploy TiDB Operator first and then deploy Diag.

4. (Optional) Set a persistent volume.

   This step sets a data volume for Diag to persist its data. To set the volume, you can configure the `diag.volume` field with the volume type in the `${HOME}/diag-collector ↪ /values-diag-collector.yaml` file. The following examples are PVC and Host:

```
# Use PVC volume type
volume:
  persistentVolumeClaim:
    claimName: local-storage-diag
```

```
# Use Host volume type
volume:
  hostPath:
    path: /data/diag
```

> **Note:**
>
> - Setting a volume on multiple disks is not supported.
> - All types of StorageClass are supported.

5. (Optional) Upgrade Diag.

   To upgrade Diag, modify the `${HOME}/diag-collector/values-diag-collector. ↪ yaml` file and then run the following command.

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

If your cluster cannot access the Internet, you can deploy Diag using the offline method.

1. Download the Diag chart.

   If your cluster cannot access the Internet, you cannot install Diag and other components by configuring the Helm repo. In this situation, you need to download the chart files on a machine with Internet access and then copy the file to the cluster.

   To download Diag chart files, you can use the following command:

   ```
   wget http://charts.pingcap.org/diag-${chart_version}.tgz
   ```

   Copy `diag-${chart_version}.tgz` to the cluster and unpack it to the current directory.

   ```
   tar zxvf diag-${chart_version}.tgz
   ```

2. Download the Diag image.

   You need to download the Diag image on a machine that has Internet access and then use the `docker load` command to load the image to the cluster.

   The Diag image is `pingcap/diag:${chart_version}`. You can download and save the image using the following commands:

   ```
   docker pull pingcap/diag:${chart_version}
   docker save -o diag-${chart_version}.tar pingcap/diag:${chart_version}
   ```

   Then, copy the archived image to the cluster and use the `docker load` command to load the image to the cluster:

   ```
   docker load -i diag-${chart_version}.tar
   ```

3. Configure the `values-diag-collector.yaml` file.

   Modify your `clinicToken` and `clinicRegion` in the `${HOME}/diag-collector/
   ↪ values-diag-collector.yaml` file.

   Other configuration parameters such as `limits`, `requests`, and `volume` can be modified according to your needs.

> **Note:**
>
> To get the token, refer to Step 2: Log in to the Clinic Server and get an access token.

4. Install Diag.

   Install Diag using the following command:

```
helm install diag-collector ./diag --namespace=tidb-admin
```

> **Note:**
>
> The `namespace` should be the same as that of TiDB Operator. If TiDB Operator is not deployed, deploy TiDB Operator first and then deploy Diag.

5. (Optional) Set a persistent volume.

   This step sets a data volume for Diag to persist its data. To set the volume, you can configure the `diag.volume` field with the volume type in the `${HOME}/diag-collector` ↪ `/values-diag-collector.yaml` file. The following examples are PVC and Host:

```
# Use PVC volume type
volume:
  persistentVolumeClaim:
    claimName: local-storage-diag
```

```
# Use Host volume type
volume:
  hostPath:
    path: /data/diag
```

> **Note:**
>
> - Setting a volume on multiple disks is not supported.
> - All types of StorageClass are supported.

To use the least privilege deployment, do the following:

> **Note:**
>
> Least privilege deployment is to deploy Diag to the namespace of the cluster so that Diag can collect data only in that namespace but not across namespaces.

1. Check the privilege of the user.

   This deployment method creates a *Role* with the following access. The user to deploy Diag needs the corresponding permissions to create a *Role* of this type.

```
Resources                                  Non-Resource URLs Resource Names
    ↪ Verbs
---------                                  ----------------- --------------
    ↪ -----
serviceaccounts                            []                []              [get
    ↪ create delete]
deployments.apps                           []                []              [get
    ↪ create delete]
rolebindings.rbac.authorization.k8s.io     []                []              [get
    ↪ create delete]
roles.rbac.authorization.k8s.io            []                []              [get
    ↪ create delete]
secrets                                    []                []              [get
    ↪ list create delete]
services                                   []                []              [get
    ↪ list create delete]
pods                                       []                []              [get
    ↪ list]
tidbclusters.pingcap.com                   []                []              [get
    ↪ list]
tidbmonitors.pingcap.com                   []                []              [get
    ↪ list]
```

2. Deploy Diag using the following `helm` command, and the latest Diag image is pulled from the Docker Hub.

```
helm install --namespace tidb-cluster diag-collector pingcap/diag --
    ↪ version ${chart_version} \
    --set diag.clinicToken=${clinic_token} \
    --set diag.clusterRoleEnabled=false \
    --set diag.clinicRegion=US
```

   If TLS is not enabled in the cluster, you can add the `--set diag.tls.enabled=false` flag, then the created *Role* will not have the `get` and `list` privileges of `secrets`.

```
helm install --namespace tidb-cluster diag-collector pingcap/diag --
    ↪ version ${chart_version} \
    --set diag.clinicToken=${clinic_token} \
    --set diag.tlsEnabled=false \
    --set diag.clusterRoleEnabled=false \
    --set diag.clinicRegion=US
```

The output is as follows:

```
NAME: diag-collector
LAST DEPLOYED: Tue Mar 15 13:00:44 2022
NAMESPACE: tidb-cluster
STATUS: deployed
REVISION: 1
NOTES:
Make sure diag-collector components are running:
  kubectl get pods --namespace tidb-cluster -l app.kubernetes.io/
    ↪ instance=diag-collector
  kubectl get svc --namespace tidb-cluster -l app.kubernetes.io/name=
    ↪ diag-collector
```

8.5.2.4 Step 4: Check the status of the Diag Pod

You can check the status of the Diag Pod using the following command:

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

The output is as follows when the Pod is running properly:

```
NAME                            READY  STATUS   RESTARTS  AGE
diag-collector-5c9d8968c-clnfr 1/1  Running  0         89s
```

8.5.3 Use Diag to collect data

You can use Diag to quickly collect diagnostic data from TiDB clusters, including monitoring data and configurations.

8.5.3.1 Usage scenarios for Diag

Diag is suitable for the following scenarios:

- When your cluster has some problems, if you need to contact PingCAP technical support, you can use Diag to collect the diagnostic data to facilitate remote troubleshooting.
- Use Diag to collect and save the data for later analysis.

> **Note:**
>
> Currently, Diag **does not support** collecting logs, configuration files, and system hardware information from clusters deployed using TiDB Operator.

8.5.3.2 Step 1: Check the data to be collected

For a full list of data that can be collected by Diag, see Clinic diagnostic Data. It is recommended to collect all data to improve the efficiency of the diagnosis.

8.5.3.3 Step 2: Collect data

You can collect data using Diag APIs.

- For detailed API documents, visit `http://${host}:${port}/api/v1`.
- To get the IP of the node, use the following command:

```
kubectl get node | grep node
```

- To get the port of `diag-collector service`, use the following command:

```
kubectl get service -n tidb-admin
```

The output is as follows:

```
NAME                TYPE          CLUSTER-IP         EXTERNAL-IP  PORT(S)
    ↪             AGE
diag-collector NodePort  10.111.143.227 <none>          4917:31917/TCP
    ↪ 18m
```

In the preceding output:

  - The port to access `diag-collector service` from outside is `31917`.
  - The service type is NodePort. You can access this service from any host in the Kubernetes cluster with its IP address `${host}` and port `${port}`.
  - If there are network restrictions between hosts, you can use the `port-forward` command to redirect the service port `4917` to local, and then use `127.0.0.1:4917` to access this service.

The following describes how to collect data using Diag APIs.

1. Request for collecting data.

   You can request for collecting data using the following API:

```
curl -s http://${host}:${port}/api/v1/collectors -X POST -d '{"
    ↪ clusterName": "${cluster-name}","namespace": "${cluster-namespace
    ↪ }","from": "2022-02-08 12:00 +0800","to": "2022-02-08 18:00 +0800
    ↪ "}'
```

   The usage of the API parameters is as follows:

- `clusterName`: the name of the TiDB cluster.
- `namespace`: the namespace name of the TiDB cluster (not the `namespace` of TiDB Operator).
- `collector`: optional, which controls the data types to be collected. The supported values include `monitor`, `config`, and `perf`. If the parameter is not specified, `monitor` and `config` data is collected by default.
- `from` and `to`: specify the start time and end time of the data collection. `+0800` indicates the time zone is UTC+8. The supported time formats are as follows:

```
"2006-01-02T15:04:05Z07:00"
"2006-01-02T15:04:05.999999999Z07:00"
"2006-01-02 15:04:05 -0700",
"2006-01-02 15:04 -0700",
"2006-01-02 15 -0700",
"2006-01-02 -0700",
"2006-01-02 15:04:05",
"2006-01-02 15:04",
"2006-01-02 15",
"2006-01-02",
```

An example output is as follows:

```
"clusterName": "${cluster-namespace}/${cluster-name}",
"collectors"         "config",
    "monitor"
],
"date": "2021-12-10T10:10:54Z",
"from": "2021-12-08 12:00 +0800",
"id": "fMcXDZ4hNzs",
"status": "accepted",
"to": "2021-12-08 18:00 +0800"
```

Descriptions of the preceding output:

- `date`: the time when the collection task is requested.
- `id`: the ID of the collection task. It is the only information to identify the collection task in the following operations.
- `status`: the current status of the task and `accepted` means the task is queued.

> **Note:**
>
> The response of the API indicates that the collection task is started but might not be completed. To check whether the collection task is completed, go to the next step.

2. Check the status of collecting data.

   To check the status of the collection task, use the following API:

```
curl -s http://${host}:${port}/api/v1/collectors/${id}
{
           "clusterName": "${cluster-namespace}/${cluster-name}",
       "collectors": [
           "config",
           "monitor"
       ],
       "date": "2021-12-10T10:10:54Z",
       "from": "2021-12-08 12:00 +0800",
       "id": "fMcXDZ4hNzs",
       "status": "finished",
       "to": "2021-12-08 18:00 +0800"
}
```

   In the preceding command, `id` is the ID of the collection task, which is `fMcXDZ4hNzs` in this case. The output format of this step is the same as the request for collecting data step.

   When the status of the collection task becomes `finished`, the collection task is completed.

3. View the collected data.

   After the collection task, you can get the collection time and data size using the following API:

```
curl -s http://${host}:${port}/api/v1/data/${id}
{
       "clusterName": "${cluster-namespace}/${cluster-name}",
       "date": "2021-12-10T10:10:54Z",
       "id": "fMcXDZ4hNzs",
       "size": 1788980746
}
```

   With the preceding command, you can **only** get the size of the dataset but cannot view the detailed data.

8.5.3.4 Step 3: Upload data

To provide cluster diagnostic data to PingCAP technical support, you need to upload the data to the Clinic Server first, and then send the obtained data access link to the staff. The Clinic Server is a cloud service that stores and shares the collected data.

1. Request for an upload task.

   You can upload the collected dataset using the following API:

```
curl -s http://${host}:${port}/api/v1/data/${id}/upload -XPOST
{
       "date": "2021-12-10T11:26:39Z",
       "id": "fMcXDZ4hNzs",
       "status": "accepted"
}
```

The response of the preceding command only indicates that the upload task is started but might not be completed. To check whether the upload task is completed, go to the next step.

2. Check the status of the upload task.

   To check the status of the upload task, use the following API:

```
curl -s http://${host}:${port}/api/v1/data/${id}/upload
{
       "date": "2021-12-10T10:23:36Z",
       "id": "fMcXDZ4hNzs",
       "result": "\"https://clinic.pingcap.com/portal/#/orgs/XXXXXXXX/
          ↪ clusters/XXXXXXXX\"",
       "status": "finished"
}
```

   When the status of the upload task becomes `finished`, the upload task is completed. At this time, `result` indicates the access link of the uploaded data in the Clinic Server, which is the link you need to send to the staff.

8.5.3.5 View data locally (optional)

The collected data is stored in the `/diag/collector/diag-${id}` directory. You can view the data in the Pod using the following steps.

1. Get `diag-collector-pod-name`.

   To get the `diag-collector-pod-name`, you can execute the following command:

```
kubectl get pod --all-namespaces | grep diag
```

   An example output is as follows:

```
tidb-admin    diag-collector-69bf78478c-nvt47          1/1    Running
   ↪          0          19h
```

   In the preceding output, the name of Diag Pod is `diag-collector-69bf78478c-nvt47` and the `namespace` is `tidb-admin`.

2. View data in Pod.

   To view data in Pod, you can use the following command. You should replace `${namespace}` with the `namespace` of TiDB Operator (usually `tidb-admin`).

```
kubectl exec -n ${namespace} ${diag-collector-pod-name} -it -- sh
cd /diag/collector/diag-${id}
```

8.5.4 Use Diag to perform a quick check on the cluster

You can use PingCAP Clinic to perform a quick check on cluster health. It mainly checks the configurations for unreasonable configuration items.

8.5.4.1 How to use

The following introduces how to use PingCAP Clinic to perform a quick check on a cluster deployed using TiDB Operator.

1. Collect data.

   For more about how to collect data, see Use Diag to collect data.

2. Diagnose data.

   You can diagnose the data locally using the following command:

```
curl -s http://${host}:${port}/api/v1/data/${id}/check -XPOST -d '{"types": ["config"]}'
```

   In the preceding output, `id` is the ID of the collection task, which is `fMcXDZ4hNzs` in this case.

   The result lists potential risks found in configurations and detailed configuration suggestions with corresponding knowledge base links. For example:

```
# Diagnostic result
basic 2022-02-07T12:00:00+08:00

## 1. Cluster basic Information
- Cluster ID: 7039963340562527412
- Cluster Name: basic
- Cluster Version: v5.4.0

## 2. Sampling Information
- Sample ID: fPrz0RnDxRn
- Sampling Date: 2022-02-07T12:00:00+08:00
- Sample Content:: [monitor config]
```

```
## 3. Diagnostic results, including potential configuration problems
In this inspection, 21 rules were executed.
The results of **3** rules were abnormal and needed to be further
   ↪ discussed with support team.
The following is the details of the abnormalities.

### Configuration rules
The configuration rules are all derived from PingCAP's OnCall Service.
If the results of the configuration rules are found to be abnormal,
   ↪ they may cause the cluster to fail.
There were **3** abnormal results.

#### Rule Name: tidb-max-days
- RuleID: 100
- Variation: TidbConfig.log.file.max-days
- For more information, please visit: https://s.tidb.io/msmo6awg
- Check Result:
TidbConfig_172.20.21.213:4000 TidbConfig.log.file.max-days:0 warning

#### Rule Name: pdconfig-max-days
- RuleID: 209
- Variation: PdConfig.log.file.max-days
- For more information, please visit: https://s.tidb.io/jkdqxudq
- Check Result:
PdConfig_172.20.22.100:2379 PdConfig.log.file.max-days:0 warning
PdConfig_172.20.14.102:2379 PdConfig.log.file.max-days:0 warning
PdConfig_172.20.15.222:2379 PdConfig.log.file.max-days:0 warning

#### Rule Name: pdconfig-max-backups
- RuleID: 210
- Variation: PdConfig.log.file.max-backups
- For more information, please visit: https://s.tidb.io/brd9zy53
- Check Result:
PdConfig_172.20.22.100:2379 PdConfig.log.file.max-backups:0 warning
PdConfig_172.20.14.102:2379 PdConfig.log.file.max-backups:0 warning
PdConfig_172.20.15.222:2379 PdConfig.log.file.max-backups:0 warning

Result report and record are saved at /diag-fPrz0RnDxRn/report
   ↪ -220208030210
```

In the preceding example:

- The first part is the basic information about the cluster.
- The second part is the sampling information.

- The third part is the diagnostic results, including potential configuration problems. For each configuration potential risk found, Diag provides a corresponding knowledge base link with detailed configuration suggestions.
- The last line is the file path of the result report and record.

9 TiDB FAQs on Kubernetes

This document collects frequently asked questions (FAQs) about the TiDB cluster on Kubernetes.

9.1 How to modify time zone settings ?

The default time zone setting for each component container of a TiDB cluster on Kubernetes is UTC. To modify this setting, take the steps below based on your cluster status:

9.1.1 For the first deployment

Configure the `.spec.timezone` attribute in the TidbCluster CR. For example:

```
...
spec:
  timezone: Asia/Shanghai
...
```

Then deploy the TiDB cluster.

9.1.2 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.

9.2 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.

9.3 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.

9.4 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.

9.5 Does TiDB Operator support TiSpark?

TiDB Operator does not yet support automatically orchestrating TiSpark.

If you want to add the TiSpark component to TiDB on 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 on Kubernetes, refer to Spark on Kubernetes.

9.6 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
```

9.7 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
```

9.8 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.

9.9 If the Ready field of a TidbCluster is false, does it mean that the corresponding TiDBCluster is unavailable?

After you execute the `kubectl get tc` command, if the output shows that the **Ready** field of a TiDBCluster is false, it does not mean that the corresponding TiDBCluster is unavailable, because the cluster might be in any of the following status:

- Upgrading
- Scaling
- Any Pod of PD, TiDB, TiKV, TiFlash, or TiProxy is not Ready

To check whether a TiDB cluster is unavailable, you can try connecting to TiDB. If the connection fails, it means that the corresponding TiDBCluster is unavailable.

9.10 After the configuration of a component is modified, why does the new configuration not take effect?

By default, after the configuration is modified, the cluster is not rolling updated and the new configuration does not take effect. To enable the automatic configuration update, you need to set `spec.configUpdateStrategy: RollingUpdate`. For details, refer to `configUpdateStrategy`.

10 Reference

10.1 Architecture

10.1.1 TiDB Operator Architecture

This document describes the architecture of TiDB Operator and how it works.

10.1.1.1 Architecture

The following diagram is an overview of the architecture of TiDB Operator.

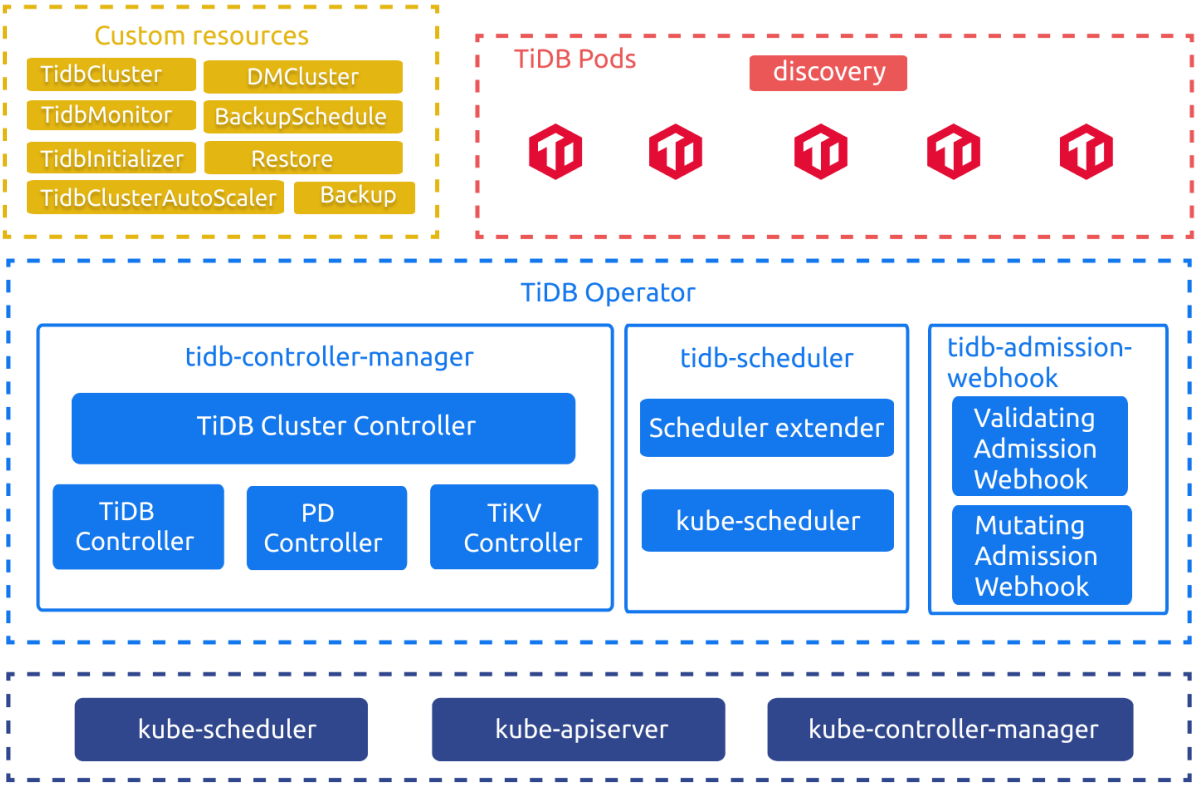


Figure 11: TiDB Operator Overview

`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.
- `discovery` is a service for inter-components discovery. Each TiDB cluster contains a discovery Pod which is used for the components to discover other existing components in the same cluster.

Note:

`tidb-scheduler` is not mandatory. Refer to tidb-scheduler and default-scheduler for details.

10.1.1.2 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.

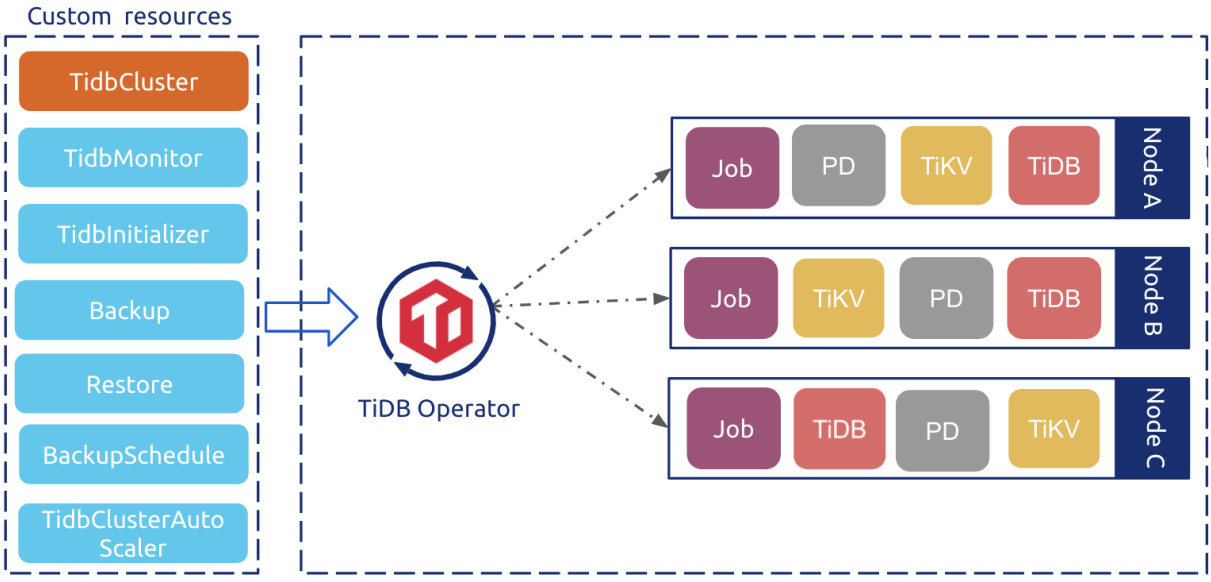


Figure 12: TiDB Operator Control Flow

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. If you configure the components to use `tidb-scheduler` in the `TidbCluster` CR, the `Pod` declaration of PD, TiKV, and TiDB specifies `tidb-scheduler` as the scheduler. `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.

10.1.2 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.

> **Note:**
>
> Starting from TiDB Operator v1.6, it is not recommended to deploy TiDB Scheduler.

10.1.2.1 tidb-scheduler and default-scheduler

A kube-scheduler is deployed by default in the Kubernetes cluster for Pod scheduling. The default scheduler name is `default-scheduler`.

In the early Kubernetes versions (< v1.16), the `default-scheduler` was not flexible enough to support even scheduling for Pods. Therefore, to support even scheduling for the TiDB cluster Pods, TiDB Operator uses a TiDB Scheduler (`tidb-scheduler`) to extend the scheduling rules of the `default-scheduler`.

Starting from Kubernetes v1.16, the `default-scheduler` has introduced the `EvenPodsSpread` feature. This feature controls how Pods are spread across your Kubernetes cluster among failure-domains. It is in the beta phase in v1.18, and became generally available in v1.19.

Therefore, if the Kubernetes cluster meets one of the following conditions, you can use `default-scheduler` directly instead of `tidb-scheduler`. You need to configure `topologySpreadConstraints` to make Pods evenly spread in different topologies.

- The Kubernetes version is v1.18.x and the `EvenPodsSpread` feature gate is enabled.
- The Kubernetes version >= v1.19.

> **Note:**
>
> When you change an existing TiDB clusters from using `tidb-scheduler` to using `default-scheduler`, it triggers the rolling update of the TiDB clusters.

10.1.2.2 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. If the `EvenPodsSpread` feature gate is not enabled in the Kubernetes cluster, 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
```

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.

10.1.2.2.1 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
…	

10.1.2.2.2 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
…		

10.1.2.3 How TiDB Scheduler works

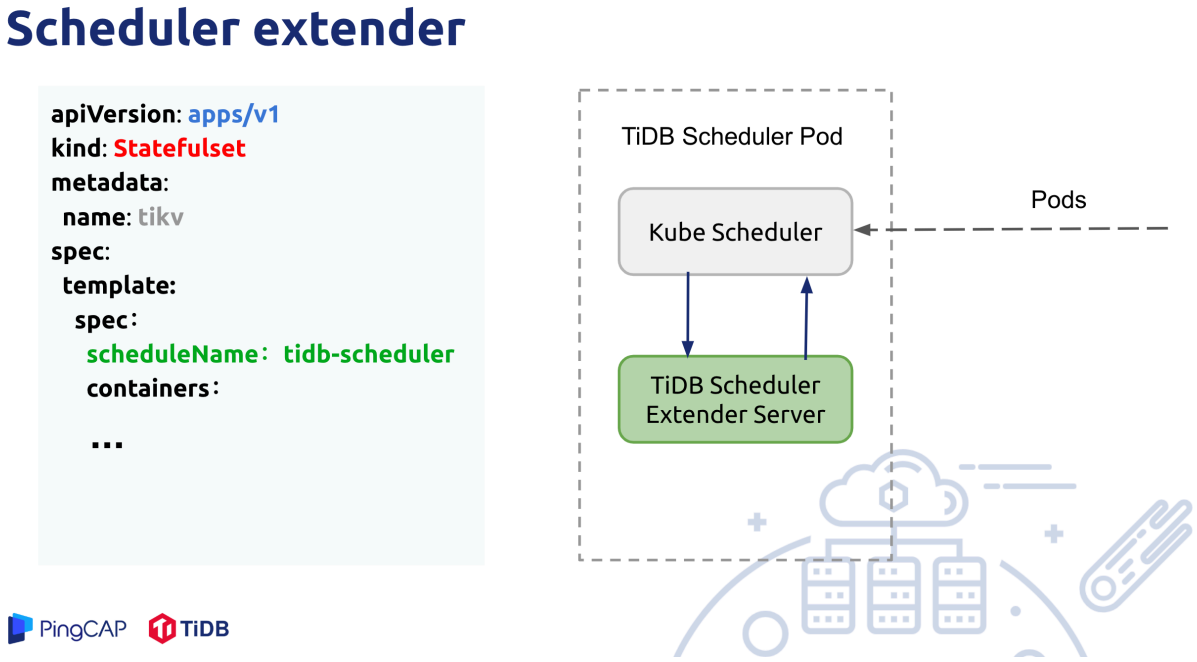


Figure 13: TiDB Scheduler Overview

TiDB Scheduler adds customized scheduling rules by implementing Kubernetes Scheduler extender.

The TiDB Scheduler component is deployed as one or more Pods, though only one Pod is working at the same time. Each Pod has two Containers inside: one Container is a native `kube-scheduler`, and the other is a `tidb-scheduler` implemented as a Kubernetes scheduler extender.

If you configure the cluster to use `tidb-scheduler` in the `TidbCluster` CR, the `.spec ↪ .schedulerName` attribute of PD, TiDB, and TiKV Pods created by TiDB Operator is set to `tidb-scheduler`. This means that the TiDB Scheduler is used for the scheduling.

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.

10.1.3 Advanced StatefulSet Controller

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.

10.1.3.1 Enable

1. Load the Advanced StatefulSet CRD file:

```
kubectl apply -f https://raw.githubusercontent.com/pingcap/tidb-
    ↪ operator/v1.6.1/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.

10.1.3.2 Usage

This section describes how to use the advanced StatefulSet controller.

10.1.3.2.1 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
```

10.1.3.2.2 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: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    maxFailoverCount: 0
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
```

```
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 4
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    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: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
```

```
  baseImage: pingcap/pd
  maxFailoverCount: 0
  replicas: 3
  requests:
    storage: "1Gi"
  config: {}
tikv:
  baseImage: pingcap/tikv
  maxFailoverCount: 0
  replicas: 3
  requests:
    storage: "1Gi"
  config: {}
tidb:
  baseImage: pingcap/tidb
  maxFailoverCount: 0
  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]`.

10.1.3.2.3 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: v8.5.0
  timezone: UTC
  pvReclaimPolicy: Delete
  pd:
    baseImage: pingcap/pd
    maxFailoverCount: 0
    replicas: 3
    requests:
      storage: "1Gi"
    config: {}
  tikv:
    baseImage: pingcap/tikv
    maxFailoverCount: 0
    replicas: 4
    requests:
      storage: "1Gi"
    config: {}
  tidb:
    baseImage: pingcap/tidb
    maxFailoverCount: 0
    replicas: 2
    service:
      type: ClusterIP
    config: {}
```

The delete-slots annotations can be left empty or deleted completely.

10.1.4 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.

10.1.4.1 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.

10.1.4.2 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
   ```

2. Configure the failure policy.

   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 on Kubernetes.

10.1.4.3 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.

10.1.4.4 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 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 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
```

10.2 TiDB on Kubernetes Sysbench Performance Test

Since the release of TiDB Operator GA, more users begin to deploy and manage the TiDB cluster on 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 on Kubernetes.

10.2.1 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

10.2.2 Test environment

10.2.2.1 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. Components not configured 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
```

10.2.2.2 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;
```

10.2.2.3 Hardware recommendations

10.2.2.3.1 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 a single AZ, the c2 machine is not simultaneously available in three AZs within the same Google Cloud 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 Google Cloud regions. In the test, the disk also performs differently. Therefore, only the machines in us-central1 are applied for test.

10.2.2.3.2 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]
```

10.2.2.3.3 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.

10.2.2.3.4 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 Google Cloud region, so n1-standard-16 machine type is chosen.

10.2.2.4 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`.

10.2.2.5 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.

10.2.2.5.1 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).

10.2.2.5.2 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
```

10.2.2.5.3 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}`.

10.2.3 Test report

10.2.3.1 In single AZ

10.2.3.1.1 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:

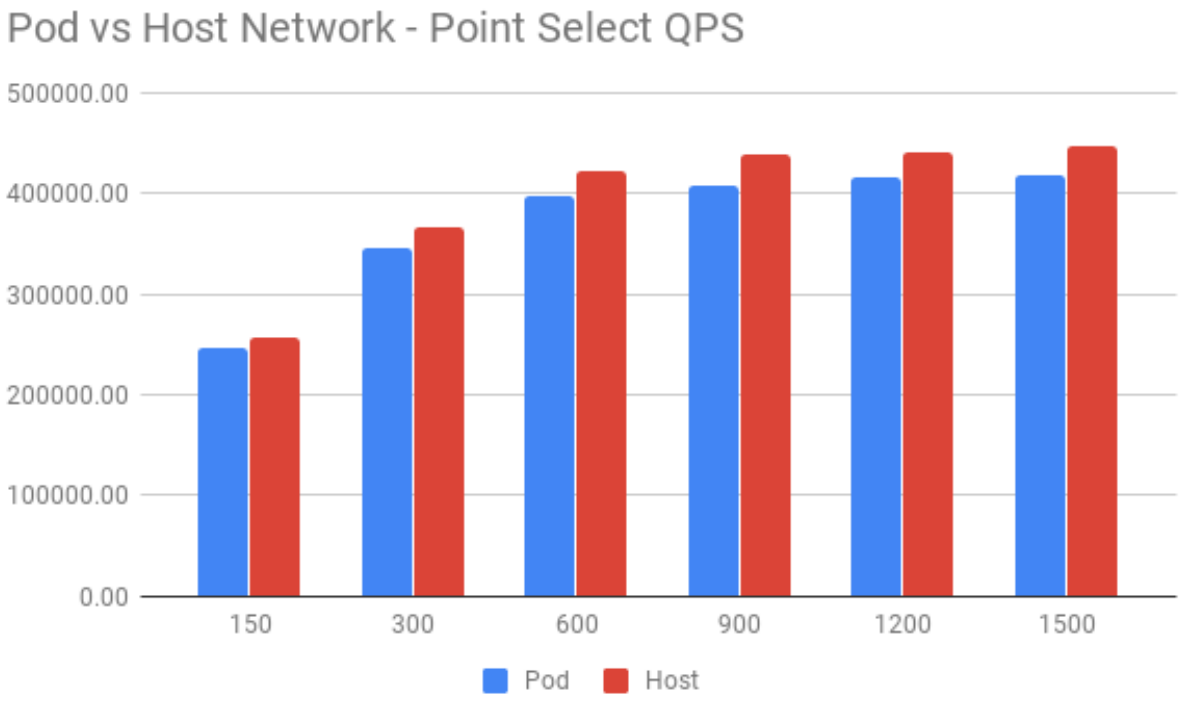


Figure 14: Pod vs Host Network

Latency comparison:

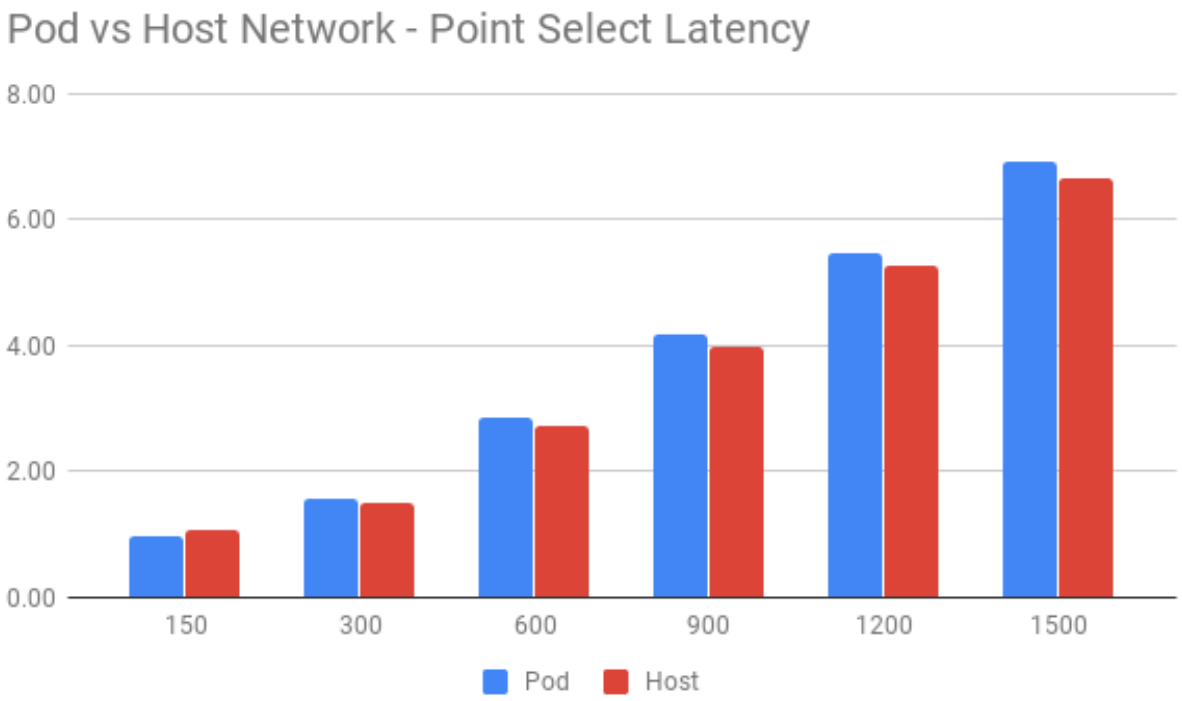


Figure 15: Pod vs Host Network

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

10.2.3.1.2 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:

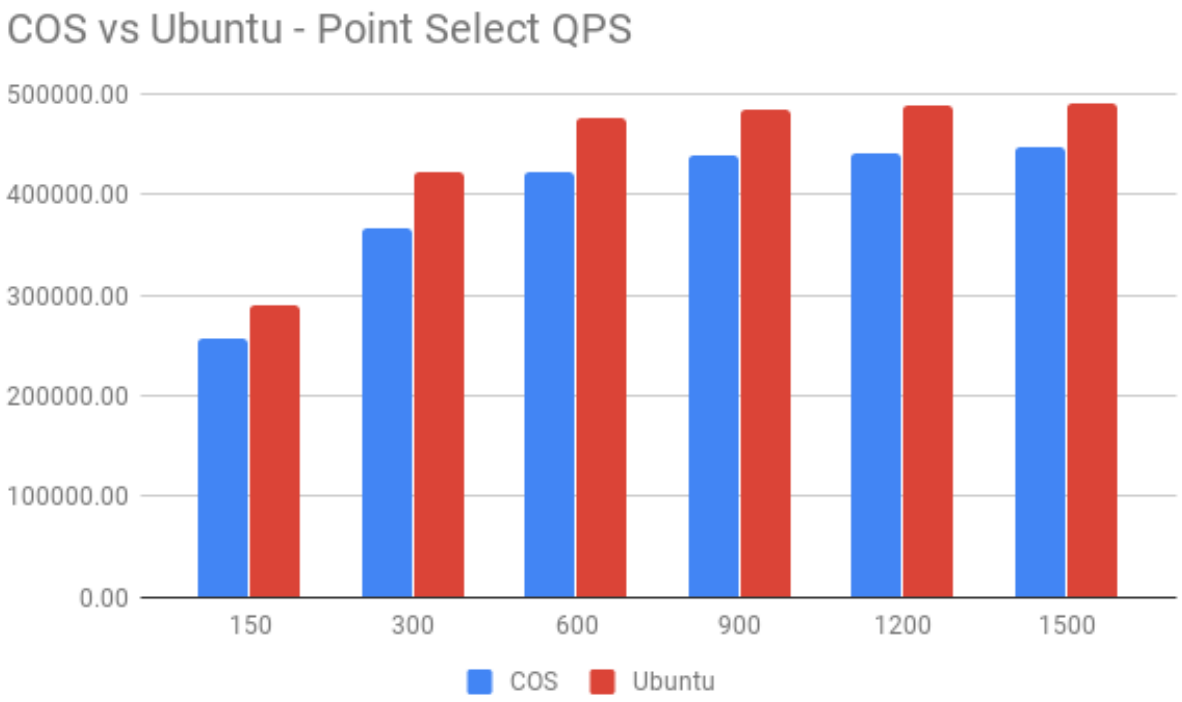


Figure 16: COS vs Ubuntu

Latency comparison:

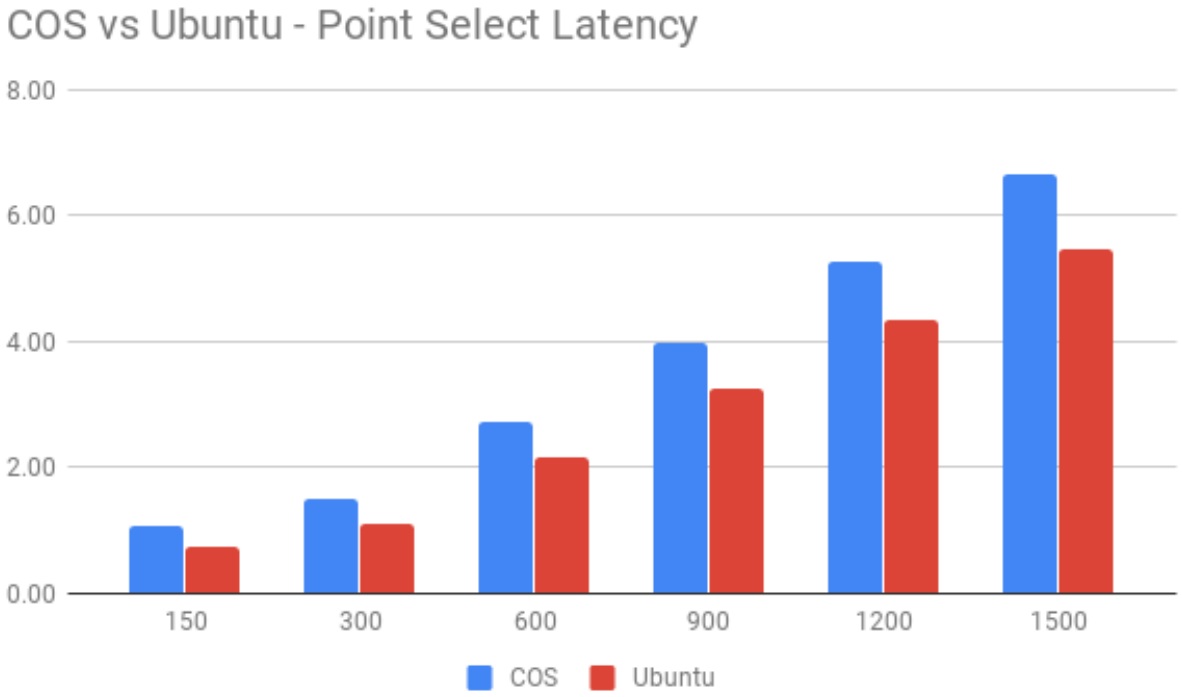


Figure 17: COS vs Ubuntu

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.

10.2.3.1.3 Kubernetes Service vs Google Cloud 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:

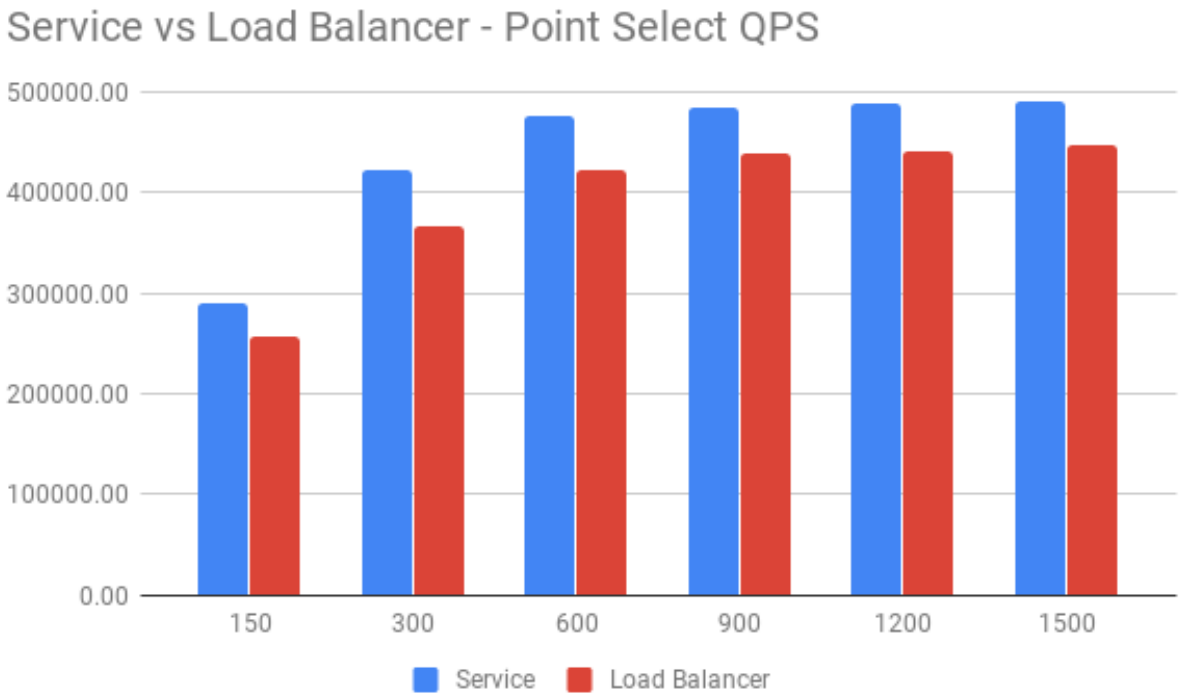


Figure 18: Service vs Load Balancer

Latency comparison:

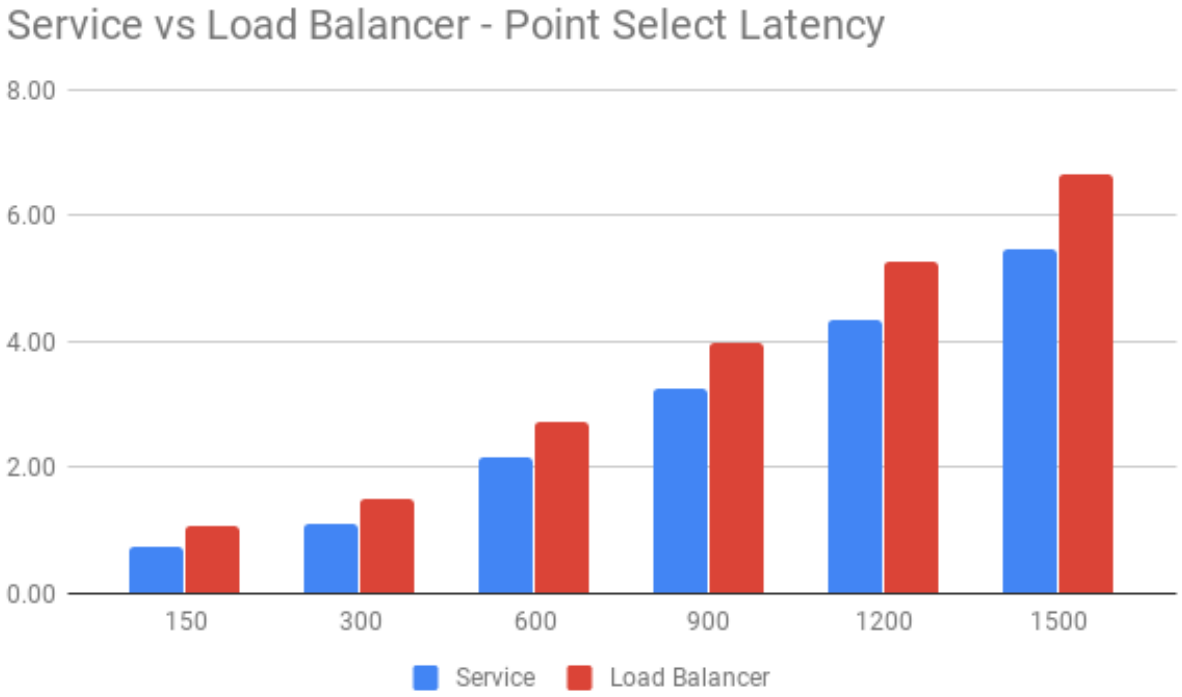


Figure 19: Service vs Load Balancer

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

10.2.3.1.4 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:

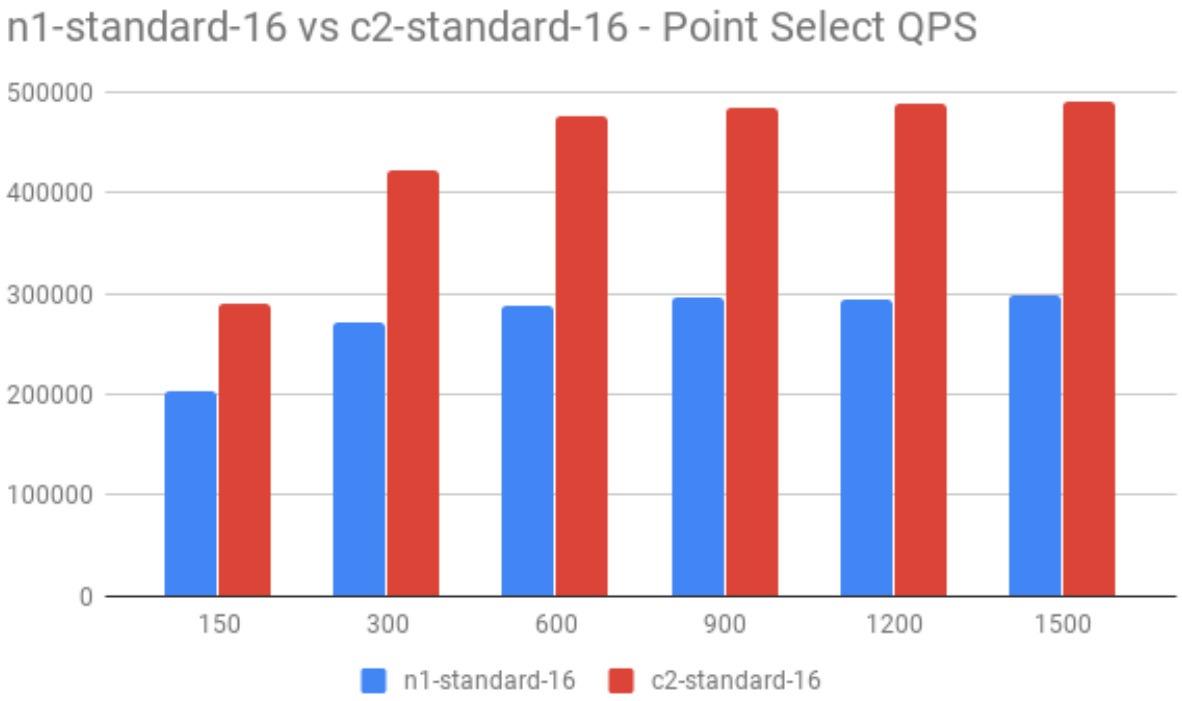


Figure 20: n1-standard-16 vs c2-standard-16

Latency comparison:

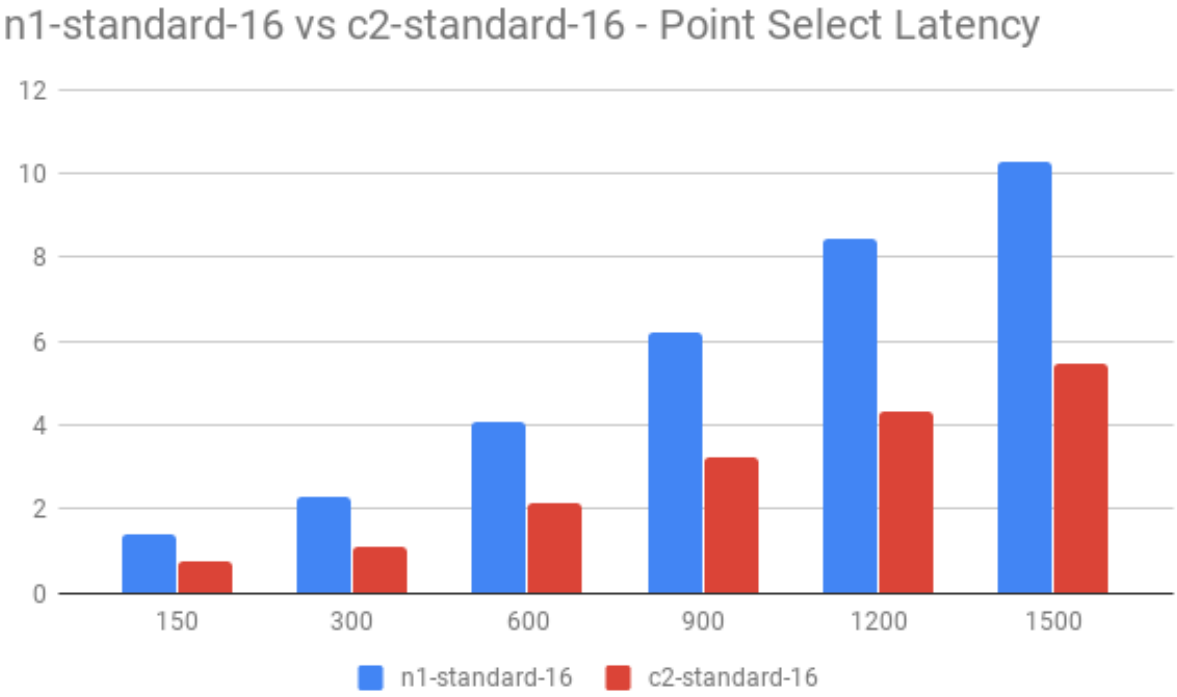


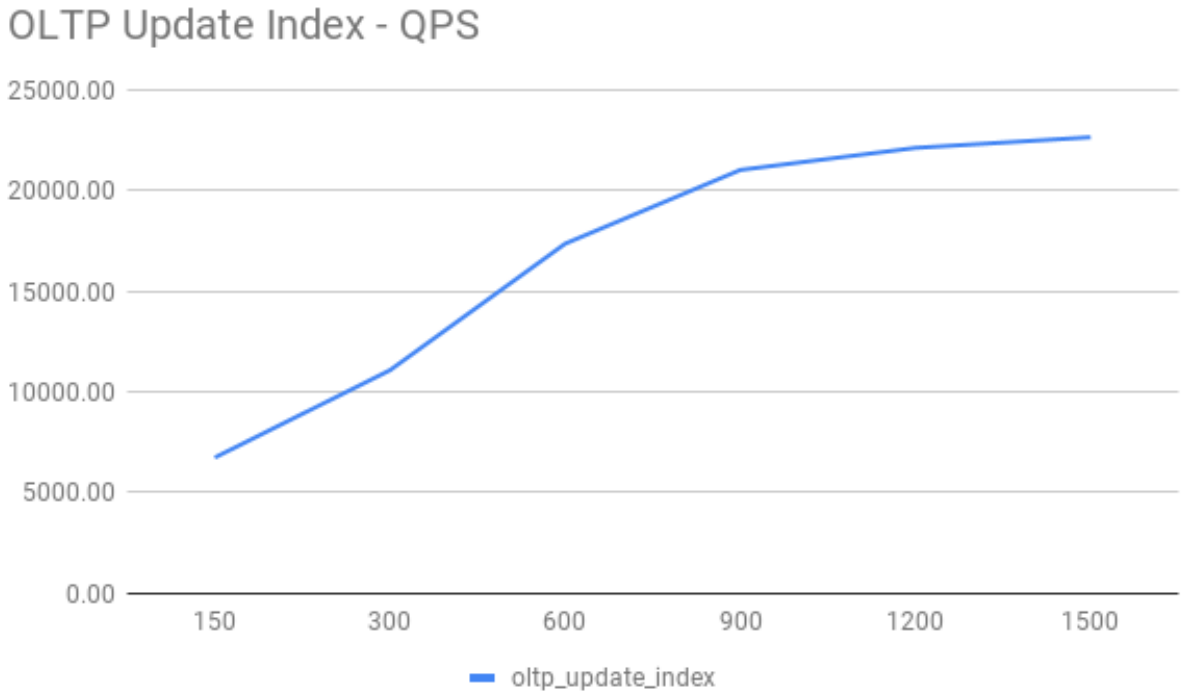
Figure 21: n1-standard-16 vs c2-standard-16

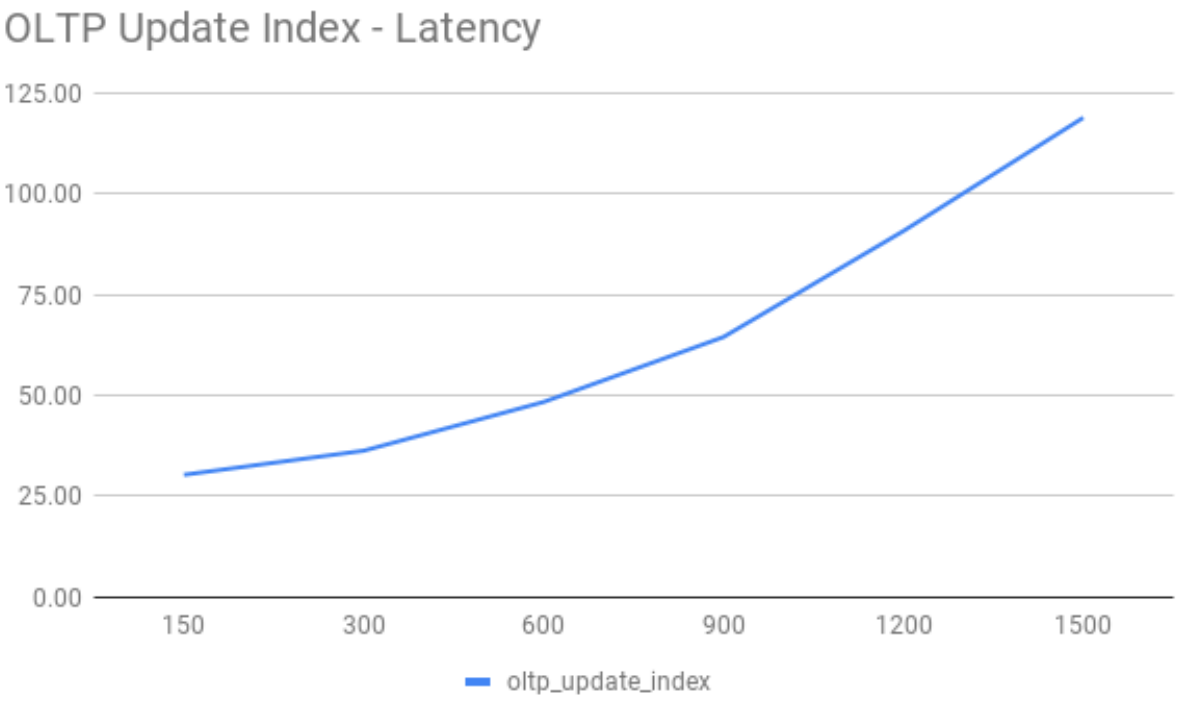
10.2.3.2 OLTP and other tests

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.

10.2.3.2.1 OLTP Update Index

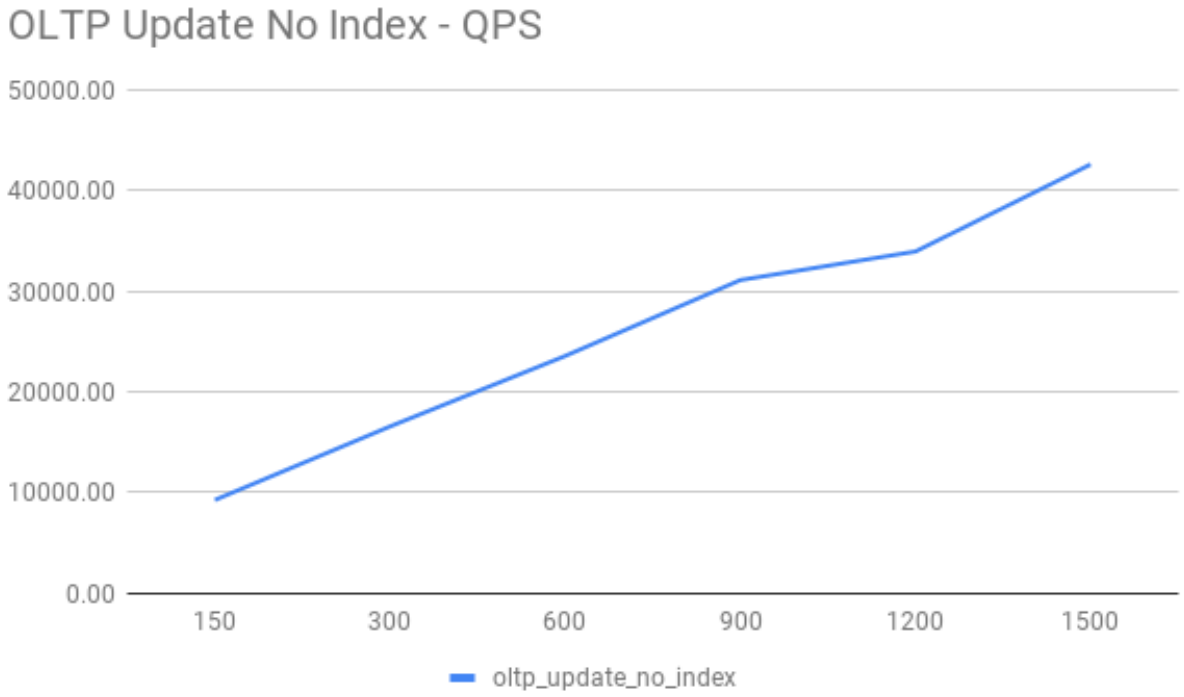
Threads	QPS	95% latency(ms)
150	6726.59	30.26
300	11067.55	36.24
600	17358.46	48.34
900	21025.23	64.47
1200	22121.87	90.78
1500	22650.13	118.92

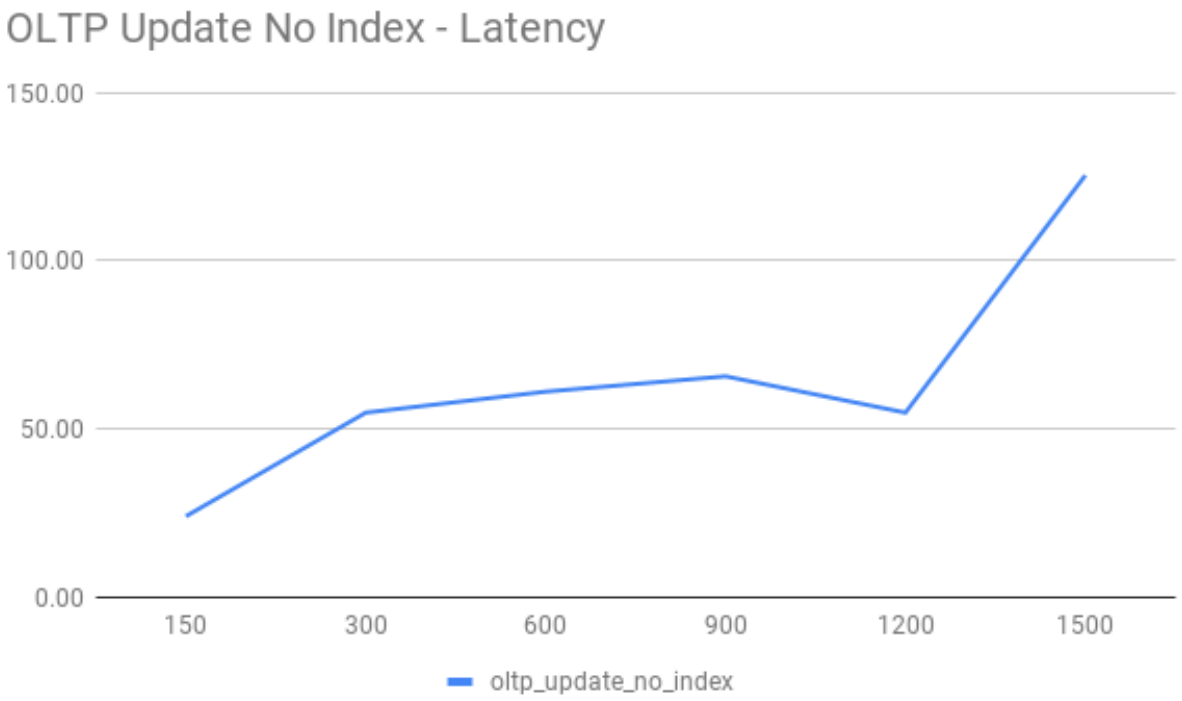




10.2.3.2.2 OLTP Update Non Index

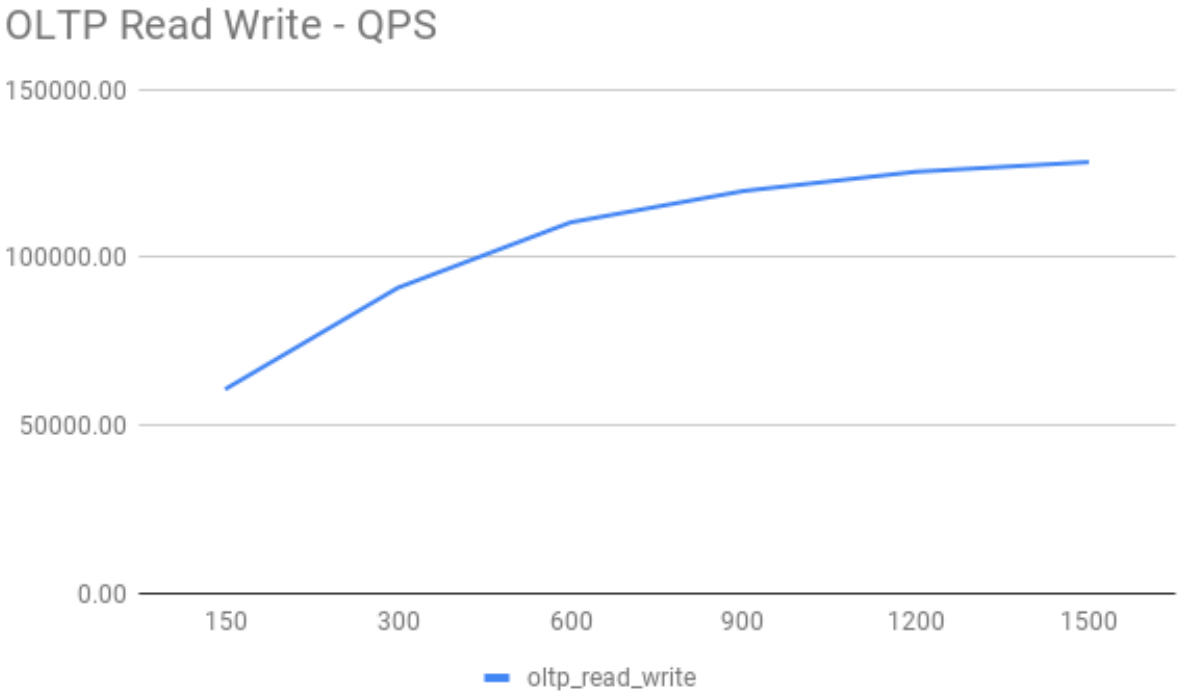
Threads	QPS	95% latency(ms)
150	9230.60	23.95
300	16543.63	54.83
600	23551.01	61.08
900	31100.10	65.65
1200	33942.60	54.83
1500	42603.13	125.52

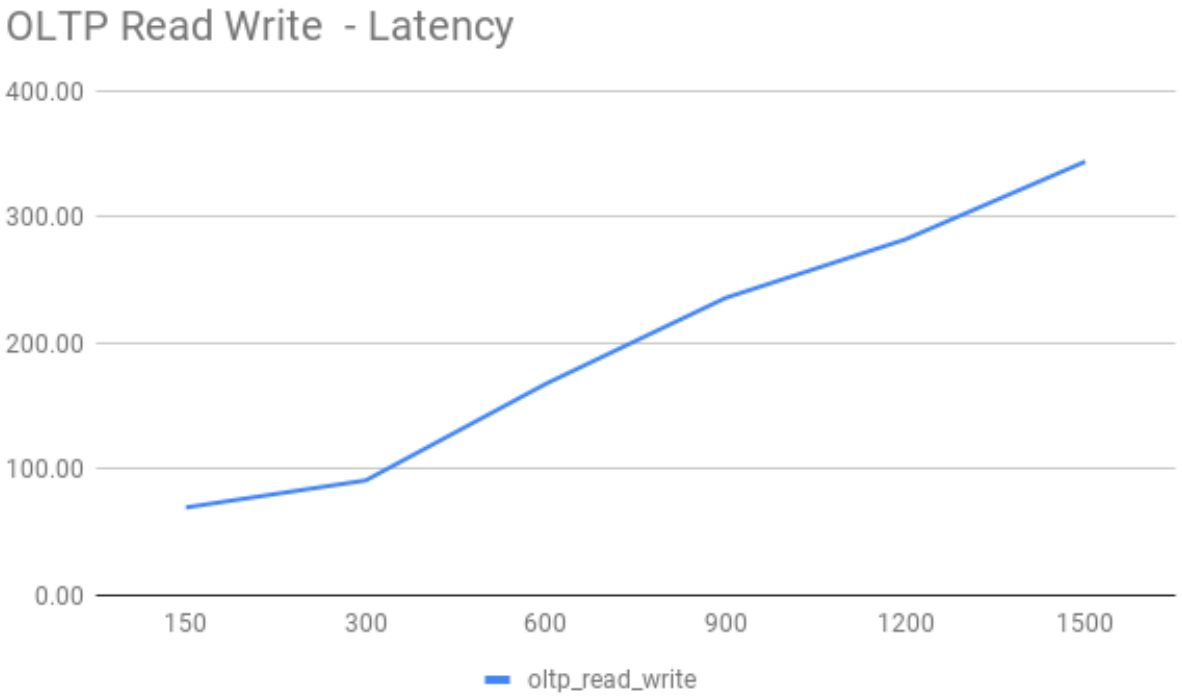




10.2.3.2.3 OLTP Read Write

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





10.2.3.3 Performance comparison between single AZ and multiple AZs

The network latency on communication across multiple AZs in Google Cloud 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:

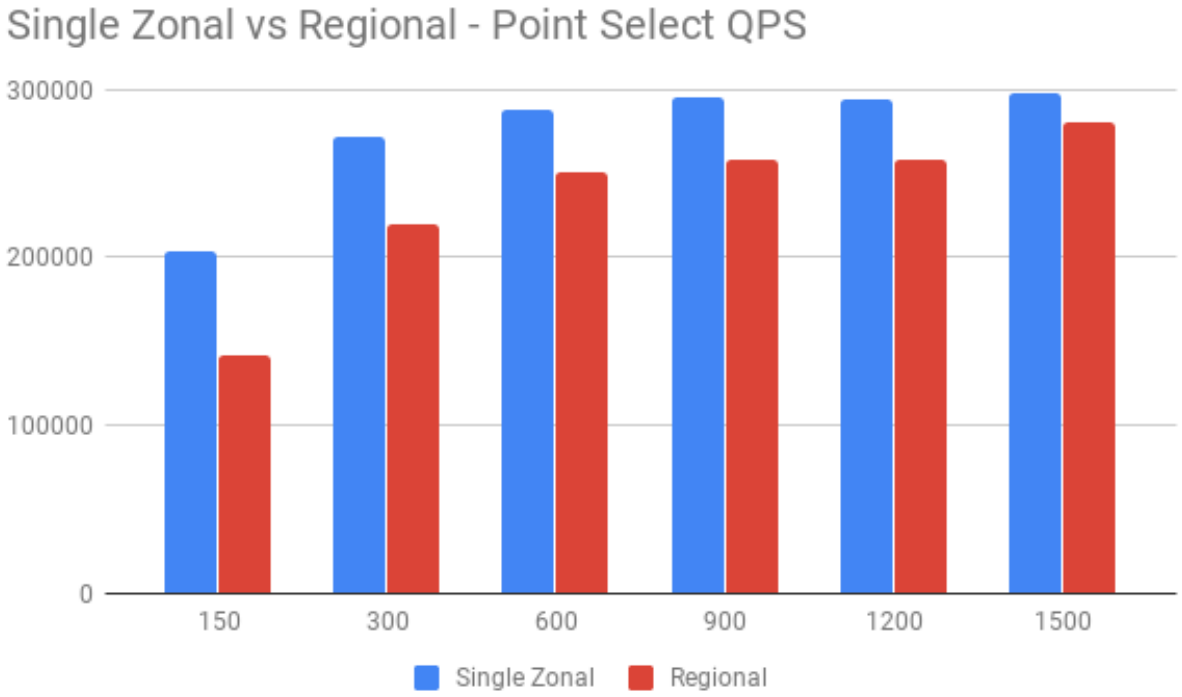


Figure 22: Single Zonal vs Regional

Latency comparison:

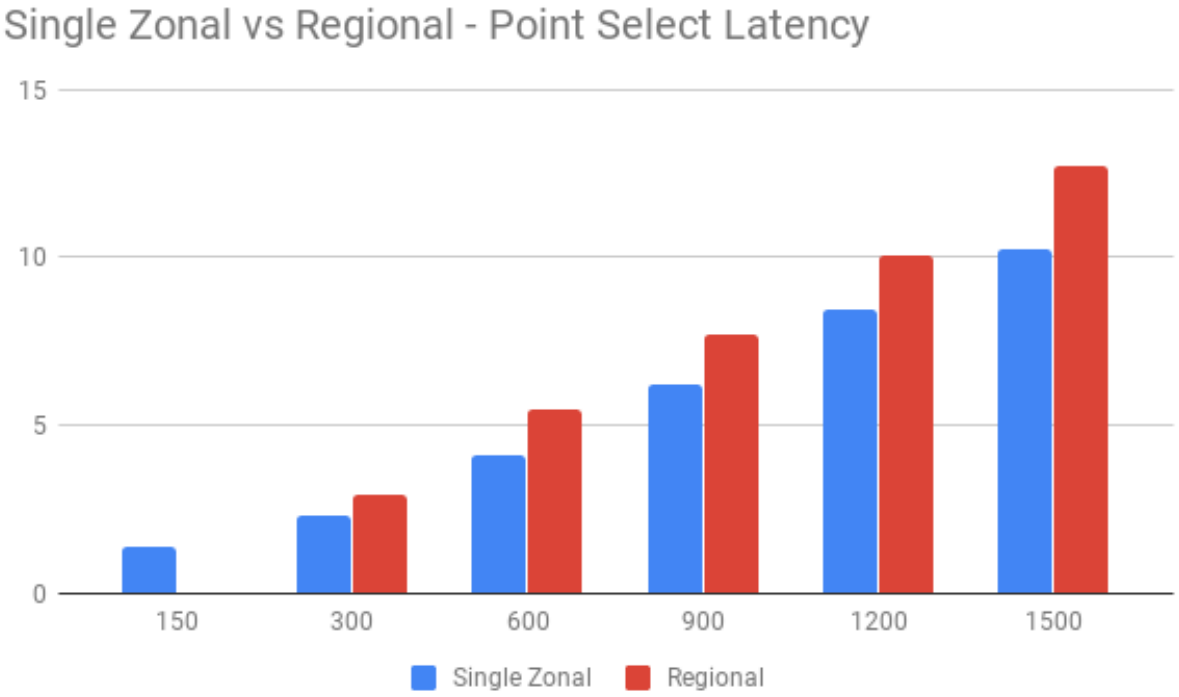


Figure 23: Single Zonal vs Regional

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.

10.2.4 Conclusion

This is a test of TiDB using sysbench running on 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 Google Cloud 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).

10.3 API References

10.4 Command Cheat Sheet for TiDB Cluster Management

This document is an overview of the commands used for TiDB cluster management.

10.4.1 kubectl

10.4.1.1 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}
```

10.4.1.2 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
```

10.4.1.3 Edit resources

- Edit TidbCluster:

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

10.4.1.4 Patch Resources

- Patch TidbCluster:

```
kubectl -n ${namespace} patch tc ${name} --type merge -p '${json_path}'
```

- 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}'
```

10.4.1.5 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}
```

10.4.1.6 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
```

10.4.1.7 Interact with nodes

- Mark the node as non-schedulable:

```
kubectl cordon ${node_name}
```

- Mark the node as schedulable:

```
kubectl uncordon ${node_name}
```

10.4.1.8 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}
```

10.4.1.9 More

See kubectl Cheat Sheet for more kubectl usage.

10.4.2 Helm

10.4.2.1 Add Helm repository

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

10.4.2.2 Update Helm repository

```
helm repo update
```

10.4.2.3 View available Helm chart

- View charts in Helm Hub:

```
helm search hub ${chart_name}
```

For example:

```
helm search hub mysql
```

- View charts in other repositories:

```
helm search repo ${chart_name} -l --devel
```

For example:

```
helm search repo tidb-operator -l --devel
```

10.4.2.4 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=v1.6.1 > values-tidb-
    ↪ operator.yaml
```

10.4.2.5 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=v1.6.1 -f values-tidb-operator.yaml
```

10.4.2.6 View the deployed Helm release

```
helm ls
```

10.4.2.7 Update Helm release

```
helm upgrade ${name} ${chart_name} --version=${chart_version} -f ${
    ↪ values_file}
```

For example:

```
helm upgrade tidb-operator pingcap/tidb-operator --version=v1.6.1 -f values-
    ↪ tidb-operator.yaml
```

10.4.2.8 Delete Helm release

```
helm uninstall ${name} -n ${namespace}
```

For example:

```
helm uninstall tidb-operator -n tidb-admin
```

10.4.2.9 More

See Helm Commands for more Helm usage.

10.5 RBAC rules required by TiDB Operator

The role-based access control (RBAC) rules implemented on Kubernetes use Role or ClusterRole for management, and use RoleBinding or ClusterRoleBinding to grant permissions to a user or a group of users.

10.5.1 Manage TiDB clusters at the cluster level

If the default setting `clusterScoped=true` is unchanged during the TiDB Operator deployment, TiDB Operator manages all TiDB clusters within a Kubernetes cluster.

To check the ClusterRole created for TiDB Operator, run the following command:

```
kubectl get clusterrole | grep tidb
```

The example output is as follows:

```
tidb-operator:tidb-controller-manager                                  2021-05-04T13
    ↪ :08:55Z
tidb-operator:tidb-scheduler                                           2021-05-04T13
    ↪ :08:55Z
```

In the output above:

- `tidb-operator:tidb-controller-manager` is the ClusterRole created for the `tidb-controller-manager` Pod.
- `tidb-operator:tidb-scheduler` is the ClusterRole created for the `tidb-scheduler` Pod.

10.5.1.1 `tidb-controller-manager` ClusterRole permissions

The following table lists the permissions corresponding to the `tidb-controller-manager` ClusterRole.

Resource	Non-resource URLs	Resource name	Action	Explanation
events	-	-	[*]	Exports event information

Resource	Non-resource URLs	Resource name	Action	Explanation
services	-	-	[*]	Control the access of the service resources
statefulsets.apps.pingcap.com/status			[*]	Control the access of the StatefulSet resource when `AdvancedStatefulSet=true`. For more information, see Advanced StatefulSet Controller.

Resource	Non-resource URLs	Resource name	Action	Explanation
statefulsets.apps.pingcap.com			[*]	Control the access of the StatefulSet resource when `AdvancedStatefulSet =true`. For more information, see Advanced StatefulSet Controller.
controllerrevisions.apps	-		[*]	Control the version of Kubernetes StatefulSet/DaemonSet
deployments.apps	-		[*]	Control the access of the Deployment resource

Resource	Non-resource URLs	Resource name	Action	Explanation
statefulsets.apps		-	[*]	Control the access of the Statefulset resource
ingresses.extensions		-	[*]	Control the access of the Ingress resource for the monitoring system
*.pingcap.com		-	[*]	Control the access of all customized resources under pingcap.com
configmaps	-	-	[create get list watch update delete]	Control the access of the ConfigMap resource

Resource	Non-resource URLs	Resource name	Action	Explanation
endpoints	-	-	[create get list watch update delete]	Control the access of the Endpoints resource
serviceaccounts	-	-	[create get update delete]	Create ServiceAccount for the Tidb-Monitor/Discovery service
clusterrolebindings.rbac.authorization.k8s.io	-	-	[create get update delete]	Create ClusterRoleBinding for the Tidb-Monitor service
rolebindings.rbac.authorization.k8s.io	-	-	[create get update delete]	Create RoleBinding for the Tidb-Monitor/Discovery service
secrets	-	-	[create update get list watch delete]	Control the access of the Secret resource

Resource	Non-resource URLs	Resource name	Action	Explanation
clusterroles.rbac.authorization.k8s.io			[escalate create get update delete]	Create ClusterRole for the Tidb-Monitor service
roles.rbac.authorization.k8s.io			[escalate create get update delete]	Create Role for the Tidb-Monitor/Discovery service
persistentvolumeclaims	-		[get list watch create update delete patch]	Control the access of the PVC resource
jobs.batch	-	-	[get list watch create update delete]	Use jobs to perform TiDB cluster initialization, backup, and restore operations

Resource	Non-resource URLs	Resource name	Action	Explanation
persistentvolumes		-	[get list watch patch update]	Perform operations such as adding labels related to cluster information for PV and modifying `persistentVolumeReclaimPolic` ↪
pods	-	-	[get list watch update delete]	Control the access of the Pod resource
nodes	-	-	[get list watch]	Read node labels and set store labels for TiKV and TiFlash accordingly

Resource	Non-resource URLs	Resource name	Action	Explanation
storageclasses.storage.k8s.io			[get list watch]	Verify whether StorageClass supports `VolumeExpansion` ↪ before expanding PVC storage
-	[/metrics]	-	[get]	Read monitoring metrics

Note:

- In the **Non-resource URLs** column, `-` indicates that the item does not have non-resource URLs.
- In the **Resource name** column, `-` indicates that the item does not have a resource name.
- In the **Actions** column, `*` indicates that the resource supports all actions that can be performed on a Kubernetes cluster.

10.5.1.2 `tidb-scheduler` ClusterRole permissions

The following table lists the permissions corresponding to the `tidb-scheduler` ClusterRole.

Resource	Non-resource URLs	Resource name	Action	Explanation
leases.coordination.k8s.io		-	[create]	Create lease resource locks for leader election
endpoints	-	-	[delete get patch update]	Control the access of the Endpoints resource
persistentvolumeclaims		-	[get list update]	Read PVC information of PD/TiKV and update the scheduling information to the PVC label
configmaps	-	-	[get list watch]	Read the ConfigMap resource
pods	-	-	[get list watch]	Read Pod information
nodes	-	-	[get list]	Read node information

Resource	Non-resource URLs	Resource name	Action	Explanation
leases.coordination.k8s.io		[tidb-scheduler]	[get update]	Read and update lease resource locks for leader election
tidbclusters.pingcap.com		-	[get]	Read Tidb-cluster information

Note:

- In the **Non-resource URLs** column, - indicates that the item does not have non-resource URLs.
- In the **Resource name** column, - indicates that the item does not have a resource name.

10.5.2 Manage TiDB clusters at the namespace level

If `clusterScoped=false` is set during the TiDB Operator deployment, TiDB Operator manages TiDB clusters at the Namespace level.

- To check the ClusterRole created for TiDB Operator, run the following command:

```
kubectl get clusterrole | grep tidb
```

The output is as follows:

```
tidb-operator:tidb-controller-manager
    ↪ 2021-05-04T13:08:55Z
```

`tidb-operator:tidb-controller-manager` is the ClusterRole created for the `tidb-controller-manager` Pod.

> **Note:**
>
> During the TiDB Operator deployment, if `controllerManager.clusterPermissions.nodes`, `controllerManager.clusterPermissions.persistentvolumes`, `controllerManager.clusterPermissions.storageclasses` are all set to `false`, TiDB operator will not create this ClusterRole.

- To check the roles created for TiDB Operator, run the following command:

```
kubectl get role -n tidb-admin
```

The example output is as follows:

```
tidb-admin  tidb-operator:tidb-controller-manager      2021-05-04T13:08:55Z
tidb-admin  tidb-operator:tidb-scheduler               2021-05-04T13:08:55Z
```

In the output:

  - `tidb-operator:tidb-controller-manager` is the role created for the `tidb-controller-manager` Pod.
  - `tidb-operator:tidb-scheduler` is the role created for the `tidb-scheduler` Pod.

10.5.2.1 `tidb-controller-manager` ClusterRole permissions

The following table lists the permissions corresponding to the `tidb-controller-manager` ClusterRole.

Resource	Non-resource URLs	Resource name	Action	Explanation
persistentvolumes		-	[get list watch patch update]	Perform operations such as adding labels related to cluster information for PV and modifying `persistentVolumeReclaimPolicy` ↪
nodes	-	-	[get list watch]	Read node Labels and set store Labels for TiKV and TiFlash accordingly
storageclasses	s.storage.k8s.io	-	[get list watch]	Verify whether StorageClass supports `VolumeExpansion` ↪ before expanding PVC storage

Note:

- In the **Non-resource URLs** column, - indicates that the item does not have non-resource URLs.
- In the **Resource name** column, - indicates that the item does not have a resource name.

10.5.2.2 `tidb-controller-manager` Role permissions

The following table lists the permissions corresponding to the `tidb-controller-manager` Role.

Resource	Non-resource URLs	Resource name	Action	Explanation
events	-	-	[*]	Export event information
services	-	-	[*]	Control the access of the service resources

Resource	Non-resource URLs	Resource name	Action	Explanation
statefulsets.apps.pingcap.com/status			[*]	Control the access of the StatefulSet resource when `AdvancedStatefulSet =true`. For more information, see Advanced StatefulSet Controller.

Resource	Non-resource URLs	Resource name	Action	Explanation
statefulsets.apps.pingcap.com		-	[*]	Control the access of the StatefulSet resource when `AdvancedStatefulSet = true`. For more information, see Advanced StatefulSet Controller.
controllerrevisions.apps		-	[*]	Control the version of Kubernetes StatefulSet/DaemonSet
deployments.apps		-	[*]	Control the access of the Deployment resource

Resource	Non-resource URLs	Resource name	Action	Explanation
statefulsets.apps		-	[*]	Control the access of the Statefulset resource
ingresses.extensions		-	[*]	Control the access of the Ingress resource for the monitoring system
*.pingcap.com		-	[*]	Control the access of all customized resources under pingcap.com
configmaps	-	-	[create get list watch update delete]	Control the access of the ConfigMap resource
endpoints	-	-	[create get list watch update delete]	Control the access of the Endpoints resource

Resource	Non-resource URLs	Resource name	Action	Explanation
serviceaccounts	-	-	[create get update delete]	Create ServiceAccount for the Tidb-Monitor/Discovery service
rolebindings.rbac.authorization.k8s.io	-	-	[create get update delete]	Create ClusterRoleBinding for the Tidb-Monitor service
secrets	-	-	[create update get list watch delete]	Control the access of the Secret resource
roles.rbac.authorization.k8s.io	-	-	[escalate create get update delete]	Create Role for the Tidb-Monitor/Discovery service
persistentvolumeclaims	-	-	[get list watch create update delete patch]	Control the access of the PVC resource

Resource	Non-resource URLs	Resource name	Action	Explanation
jobs.batch	-	-	[get list watch create update delete]	Use jobs to perform TiDB cluster initialization, backup, and restore operations
pods	-	-	[get list watch update delete]	Control the access of the Pod resource

Note:

- In the **Non-resource URLs** column, `-` indicates that the item does not have non-resource URLs.
- In the **Resource names** column, `-` indicates that the item does not have a resource name.
- In the **Actions** column, `*` indicates that the resource supports all actions that can be performed on a Kubernetes cluster.

10.5.2.3 `tidb-scheduler` Role permissions

The following table lists the permissions corresponding to the `tidb-scheduler` Role.

Resource	Non-resource URLs	Resource name	Action	Explanation
leases.coordination.k8s.io		-	[create]	Create lease resource locks for leader election
endpoints	-	-	[delete get patch update]	Control the access of the Endpoints resource
persistentvolumeclaims		-	[get list update]	Read PVC information of PD/TiKV and update the scheduling information to the PVC label
configmaps	-	-	[get list watch]	Read the ConfigMap resource
pods	-	-	[get list watch]	Read pod information
nodes	-	-	[get list]	Read node information

Resource	Non-resource URLs	Resource name	Action	Explanation
leases.coordination.k8s.io		[tidb-scheduler]	[get update]	Read and update lease resource locks for leader election
tidbclusters.pingcap.com		-	[get]	Read Tidb-cluster information

Note:

- In the **Non-resource URLs** column, - indicates that the item does not have non-resource URLs.
- In the **Resource name** column, - indicates that the item does not have a resource name.

10.6 Tools

10.6.1 Tools on Kubernetes

Operations on TiDB on 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 on Kubernetes.

10.6.1.1 Use PD Control on Kubernetes

PD Control is the command-line tool for PD (Placement Driver). To use PD Control to operate on TiDB clusters on 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
```

10.6.1.2 Use TiKV Control on Kubernetes

TiKV Control is the command-line tool for TiKV. When using TiKV Control for TiDB clusters on 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 Debug 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 debug 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
```

10.6.1.3 Use TiDB Control on Kubernetes

TiDB Control is the command-line tool for TiDB. To use TiDB Control on 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
```

10.6.1.4 Use Helm

Helm is a package management tool for Kubernetes. The installation steps are as follows:

10.6.1.4.1 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 version`. 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"}
```

10.6.1.4.2 Configure the Helm repo

Kubernetes applications are packed as charts in Helm. PingCAP provides the following Helm charts for TiDB on 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;

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   v1.6.1                      A Helm chart for TiDB
    ↪ Backup or Restore
pingcap/tidb-cluster v1.6.1                       A Helm chart for TiDB
    ↪ Cluster
pingcap/tidb-drainer v1.6.1                       A Helm chart for TiDB
    ↪ Binlog drainer.
```

```
pingcap/tidb-lightning v1.6.1                    A Helm chart for TiDB
    ↪ Lightning
pingcap/tidb-operator v1.6.1       v1.6.1        tidb-operator Helm chart
    ↪ for Kubernetes
```

When a new version of chart has been released, you can use `helm repo update` to update the repository cached locally:

```
helm repo update
```

10.6.1.4.3 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.

10.6.1.4.4 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-v1.6.1.tgz
wget http://charts.pingcap.org/tidb-drainer-v1.6.1.tgz
wget http://charts.pingcap.org/tidb-lightning-v1.6.1.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.v1.6.1.tgz
helm install ${release_name} ./tidb-operator --namespace=${namespace}
```

10.6.1.5 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 on Kubernetes use Terraform to create and manage TiDB clusters on public clouds.

Follow the steps in Terraform Documentation to install Terraform.

10.7 Configure

10.7.1 TiDB Binlog Drainer Configurations on Kubernetes

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

> **Warning:**
>
> Starting from TiDB v7.5.0, TiDB Binlog replication is deprecated. Starting from v8.3.0, TiDB Binlog is fully deprecated, with removal planned for a future release. For incremental data replication, use TiCDC instead. For point-in-time recovery (PITR), use PITR.

10.7.1.1 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"
```

10.8 TiDB Log Collection on 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.

10.8.1 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.

10.8.2 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.

10.9 Monitoring and Alerts on Kubernetes

This document describes how to monitor a Kubernetes cluster and configure alerts for the cluster.

10.9.1 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.

10.9.1.1 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.

10.9.1.2 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.

10.9.1.3 Alerts on 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.

10.10 PingCAP Clinic Diagnostic Data

PingCAP Clinic Diagnostic Service (PingCAP Clinic) collects diagnostic data from TiDB clusters that are deployed using TiDB Operator. This document lists the types of data collected and their corresponding parameters.

When you collect data using Diag client (Diag), you can add the required parameters to the command according to your needs.

The diagnostic data collected by PingCAP Clinic is **only** used for troubleshooting cluster problems.

Clinic Server is a diagnostic service deployed in the cloud. There are two independent services based on different storage locations:

- Clinic Server for international users: If you upload the collected data to Clinic Server for international users, the data will be stored in the Amazon S3 set up by PingCAP in AWS US regions. PingCAP strictly controls permissions for data and only allows authorized internal technical support staff to access the data.
- Clinic Server for users in the Chinese mainland: If you upload the collected data to Clinic Server for users in the Chinese mainland, the data will be stored in the Amazon S3 set up by PingCAP in AWS China (Beijing) regions. PingCAP strictly controls permissions for data and only allows authorized internal technical support staff to access the data.

10.10.1 TiDB cluster information

Data type	Exported file	Parameter for PingCAP Clinic
Basic information of the cluster, including the cluster ID	`cluster` ↪ `.` ↪ `json` ↪	The data is collected per run by default.
Detailed information of the cluster	`tidbcluster` ↪ `.` ↪ `json` ↪	The data is collected per run by default.

10.10.2 TiDB diagnostic data

Data type	Exported file	Parameter for PingCAP Clinic
Real-time configuration	`config.json`	`collectors:config`

10.10.3 TiKV diagnostic data

Data type	Exported file	Parameter for PingCAP Clinic
Real-time configuration	`config.json`	`collectors:config`

10.10.4 PD diagnostic data

Data type	Exported file	Parameter for PingCAP Clinic
Real-time configuration	`config` ↪ `.` ↪ `json` ↪	`collectors` ↪ `:` ↪ `config` ↪

Data type	Exported file	Parameter for PingCAP Clinic
Outputs of the command `tiup ctl pd -u http://${pd IP}:${PORT} store`	`store.json`	`collectors:config`

Data type	Exported file	Parameter for PingCAP Clinic
Outputs of the command `tiup ctl pd -u http://${pd IP}:${PORT} config placement rules show`	`placement-rule.json`	`collectors:config`

10.10.5 TiFlash diagnostic data

Data type	Exported file	Parameter for PingCAP Clinic
Real-time configuration	`config.json`	`collectors:config`

10.10.6 TiCDC diagnostic data

Data type	Exported file	Parameter for PingCAP Clinic
Real-time configuration	`config` ↪ `.` ↪ `json` ↪	`collectors` ↪ `:` ↪ `config` ↪
Debug data	`info.` ↪ `txt`, `status` ↪ `.` ↪ `txt`, `changefeeds` ↪ `.` ↪ `txt`, `captures` ↪ `.` ↪ `txt`, `processors` ↪ `.` ↪ `txt`	`collectors` ↪ `:` ↪ `debug` (Diag does not collect this data type by default)

10.10.7 Prometheus monitoring data

Data type	Exported file	Parameter for PingCAP Clinic
All Metrics data	`{metric_name}.json`	`collectors:monitor`
Alert configuration	`alerts.json`	`collectors:monitor`

11 Release Notes

11.1 v1.6

11.1.1 TiDB Operator 1.6.1 Release Notes

Release date: December 25, 2024

TiDB Operator version: 1.6.1

11.1.1.1 New features

- Support authentication for backup and restore operations using Azure Blob Storage Shared Access Signature (SAS) tokens (#5720, [@tennix](https://github.com/tennix))
- The VolumeReplace feature supports the TiFlash component (#5685, [@rajsuvariya](https://github.com/rajsuvariya))
- Add a more straightforward interface for Log Backup that supports pausing and resuming backup tasks (#5710, [@RidRisR](https://github.com/RidRisR))
- Support deleting Log Backup tasks by removing their associated Backup custom resource (CR) (#5754, [@RidRisR](https://github.com/RidRisR))
- The VolumeModify feature supports modifying Azure Premium SSD v2 disks. To use this feature, you need to grant the tidb-controller-manager permission to operate Azure disk through a node or Pod. (#5958, [@handlerww](https://github.com/handlerww))

11.1.1.2 Improvements

- The VolumeModify feature no longer performs leader eviction for TiKV, reducing modification time (#5826, [@csuzhangxc](https://github.com/csuzhangxc))
- Support specifying the minimum wait time during PD Pod rolling updates by using annotations (#5827, [@csuzhangxc](https://github.com/csuzhangxc))
- The VolumeReplace feature supports customizing the number of spare replicas for PD and TiKV (#5666, [@anish-db](https://github.com/anish-db))
- The VolumeReplace feature can be enabled for specific TiDB clusters (#5670, [@rajsuvariya](https://github.com/rajsuvariya))
- Optimize the primary transfer logic of the PD microservice to reduce the number of primary transfer operations during component updates (#5643, [@HuSharp](https://github.com/HuSharp))
- Support setting `LoadBalancerClass` for the TiDB service (#5964, [@csuzhangxc](https://github.com

11.1.1.3 Bug fixes

- Fix the issue that EBS snapshot restore incorrectly succeeds when no TiKV instances are configured or TiKV replica is set to 0 (#5659, [@BornChanger](https://github.com/BornChanger)
- Fix the issue that ClusterRole and ClusterRoleBinding resources are not properly cleaned up when you delete a `TidbMonitor` that monitors multiple TiDB clusters across namespaces (#5956, [@csuzhangxc](https://github.com/csuzhangxc))
- Fix a type mismatch issue in the `.spec.prometheus.remoteWrite.remoteTimeout` field of `TidbMonitor` (#5734, [@IMMORTALxJO](https://github.com/IMMORTALxJO))

11.1.2 TiDB Operator 1.6.0 Release Notes

Release date: May 28, 2024

TiDB Operator version: 1.6.0

11.1.2.1 New features

- Support setting `maxSkew`, `minDomains`, and `nodeAffinityPolicy` in `topologySpreadConstraints` for components of a TiDB cluster (#5617, [@csuzhangxc](https://github.com/csuzhangxc))
- Support setting additional command-line arguments for TiDB components (#5624, [@csuzhangxc](https://github.com/csuzhangxc))
- Support setting `nodeSelector` for the TidbInitializer component (#5594, [@csuzhangxc](https://gith

11.1.2.2 Improvements

- Support automatically setting location labels for TiProxy (#5649, [@djshow832](https://github.com/
- Support retrying leader eviction during TiKV rolling restart (#5613, [@csuzhangxc](https://github.co
- Support setting the `advertise-addr` command-line argument for the TiProxy component (#5608, [@djshow832](https://github.com/djshow832))

11.1.2.3 Bug fixes

- Fix the issue that modifying the storage size of components might cause them to restart when `configUpdateStrategy` is set to `InPlace` (#5602, [@ideascf](https://github.com/ideascf))
- Fix the issue that recreating the TiKV StatefulSet might cause TiKV to enter the `Upgrading` phase (#5551, [@ideascf](https://github.com/ideascf))

11.1.3 TiDB Operator 1.6.0-beta.1 Release Notes

Release date: March 27, 2024

TiDB Operator version: 1.6.0-beta.1

11.1.3.1 New features

- Support deploying PD v8.0.0 and later versions in microservice mode (experimental) (#5398, [@HuSharp](https://github.com/HuSharp))
- Support scaling out or in TiDB components in parallel (#5570, [@csuzhangxc](https://github.com/csu
- Support setting `livenessProbe` and `readinessProbe` for the Discovery component (#5565, [@csuzhangxc](https://github.com/csuzhangxc))
- Support setting `startupProbe` for TiDB components (#5588, [@fgksgf](https://github.com/fgksgf))

11.1.3.2 Improvements

- Upgrade Kubernetes dependency to v1.28, and it is not recommended to deploy tidb-scheduler (#5495, [@csuzhangxc](https://github.com/csuzhangxc))

- When deploying using Helm chart, support setting lock resource used by tidb-controller-manager for leader election, with the default value of `.Values.controllerManager.leaderResourceLock: leases`. When upgrading from versions before v1.6 to v1.6.0-beta.1 and later versions, it is recommended to first set `.Values.controllerManager.leaderResourceLock: endpointsleases` and wait for the new tidb-controller-manager to run normally before setting it to `.Values.controllerManager.leaderResourceLock: leases` to update the deployment (#5450, [@csuzhangxc](https://github.com/csuzhangxc))
- Support for TiFlash to directly mount ConfigMap without relying on an InitContainer to process configuration files (#5552, [@ideascf](https://github.com/ideascf))
- Add check for `resources.request.storage` in the `storageClaims` configuration of TiFlash (#5489, [@unw9527](https://github.com/unw9527))

11.1.3.3 Bug fixes

- Fix the issue that the `tikv-min-ready-seconds` check is not performed on the last TiKV Pod during a rolling restart of TiKV (#5544, [@wangz1x](https://github.com/wangz1x))
- Fix the issue that the TiDB cluster cannot start when only non-`cluster.local` clusterDomain TLS certificates are available (#5560, [@csuzhangxc](https://github.com/csuzhangxc))

11.2 v1.5

11.2.1 TiDB Operator 1.5.5 Release Notes

Release date: January 21, 2025

TiDB Operator version: 1.5.5

11.2.1.1 New features

- Add a more straightforward interface for Log Backup that supports pausing and resuming backup tasks (#5710, [@RidRisR](https://github.com/RidRisR))
- Support deleting Log Backup tasks by removing their associated Backup custom resource (CR) (#5754, [@RidRisR](https://github.com/RidRisR))

11.2.1.2 Improvements

- The VolumeModify feature no longer performs leader eviction for TiKV, reducing modification time (#5826, [@csuzhangxc](https://github.com/csuzhangxc))
- Support specifying the minimum wait time during PD Pod rolling updates by using annotations (#5827, [@csuzhangxc](https://github.com/csuzhangxc))

11.2.2 TiDB Operator 1.5.4 Release Notes

Release date: September 13, 2024

TiDB Operator version: 1.5.4

11.2.2.1 Improvements

- The VolumeReplace feature supports customizing the number of spare replicas for PD and TiKV (#5666, [@anish-db](https://github.com/anish-db))
- The VolumeReplace feature can be enabled for specific TiDB clusters (#5670, [@rajsuvariya](https://github.com/rajsuvariya))
- EBS snapshot restore supports configuring whether to terminate the entire restore task immediately if volume warmup fails (#5622, [@michaelmdeng](https://github.com/michaelmdeng))
- When using the `check-wal-only` warmup strategy, EBS snapshot restore marks the entire restore task as failed if warmup fails (#5621, [@michaelmdeng](https://github.com/michaelmdeng

11.2.2.2 Bug fixes

- Fix the issue that `tidb-backup-manager` cannot parse backup file size in BR backupmeta v2 (#5411, [@Leavrth](https://github.com/Leavrth))
- Fix a potential EBS volume leak issue that occurs when EBS snapshot restore fails (#5634, [@WangLe1321](https://github.com/WangLe1321))
- Fix the issue that metrics are not properly initialized after federated manager restarts (#5637, [@wxiaomou](https://github.com/wxiaomou))
- Fix the issue that EBS snapshot restore incorrectly succeeds when no TiKV instances are configured or TiKV replica is set to 0 (#5659, [@BornChanger](https://github.com/BornChanger)

11.2.3 TiDB Operator 1.5.3 Release Notes

Release date: April 18, 2024

TiDB Operator version: 1.5.3

11.2.3.1 New features

- Support setting `livenessProbe` and `readinessProbe` for the Discovery component (#5565, [@csuzhangxc](https://github.com/csuzhangxc))
- Support setting `nodeSelector` for the TidbInitializer component (#5594, [@csuzhangxc](https://gith

11.2.3.2 Bug fixes

- Fix the issue that modifying the storage size of components might cause them to restart when `configUpdateStrategy` is set to `InPlace` (#5602, [@ideascf](https://github.com/ideascf))
- Fix the issue that the `tikv-min-ready-seconds` check is not performed on the last TiKV Pod during a rolling restart of TiKV (#5544, [@wangz1x](https://github.com/wangz1x))
- Fix the issue that the TiDB cluster cannot start when only non-`cluster.local` clusterDomain TLS certificates are available (#5560, [@csuzhangxc](https://github.com/csuzhangxc))

11.2.4 TiDB Operator 1.5.2 Release Notes

Release date: January 19, 2024

TiDB Operator version: 1.5.2

11.2.4.1 New features

Starting from v1.5.2, TiDB Operator supports backing up and restoring the data of a TiDB cluster deployed across multiple AWS Kubernetes clusters to AWS storage using EBS volume snapshots. For more information, refer to Back up Data Using EBS Snapshots across Multiple Kubernetes and Restore Data Using EBS Snapshots across Multiple Kubernetes. (#5003, [@BornChanger](https://github.com/BornChanger), [@WangLe1321](https://github.com/WangLe1321), [@YuJuncen](https://github.com/YuJuncen), [@csuzhangxc](https://github.com/csuzhangxc))

11.2.4.2 Improvements

- `startScriptVersion: v2` supports waiting for Pod IP to match the one published to external DNS before starting a PD or TiKV to better support scenarios such as Stale Read (#5381, [@smineyev81](https://github.com/smineyev81))
- `startScriptVersion: v2` supports explicitly specifying PD addresses to better support scenarios where a TiDB cluster is deployed across Kubernetes (#5400, [@smineyev81](https://github.com/smineyev81))
- tidb-operator Helm Chart supports specifying resource lock for leader election when deploying Advanced StatefulSet (#5448, [@csuzhangxc](https://github.com/csuzhangxc))

11.2.4.3 Bug fixes

- Fix the issue that changing meta information such as annotations and replacing volumes at the same time might cause a deadlock for TiDB Operator reconcile (#5382, [@anish-db](https://github.com/anish-db))
- Fix the issue that the PD member might be set to the wrong label when replacing volumes (#5393, [@anish-db](https://github.com/anish-db))

11.2.5 TiDB Operator 1.5.1 Release Notes

Release date: October 20, 2023

TiDB Operator version: 1.5.1

11.2.5.1 New features

- Support replacing volumes for PD, TiKV, and TiDB (#5150, [@anish-db](https://github.com/anish-db))

11.2.5.2 Improvements

11.2.5.3 Bug fixes

- Fix errors from PVC modifier when a manual TiKV eviction is requested (#5302, [@anish-db](https://github.com/anish-db))
- Fix a deadlock issue in TiDB Operator caused by the TiKV eviction when a volume replacement is ongoing (#5301, [@anish-db](https://github.com/anish-db))
- Fix the issue that TidbCluster may be able to roll back during the upgrade process (#5345, [@anish-db](https://github.com/anish-db))
- Fix the issue that the `MaxReservedTime` option does not work for scheduled backup (#5148, [@BornChanger](https://github.com/BornChanger))

11.2.6 TiDB Operator 1.5.0 Release Notes

Release date: August 4, 2023

TiDB Operator version: 1.5.0

11.2.6.1 Rolling update changes

If TiFlash is deployed in a TiDB cluster that is v7.1.0 or later, the TiFlash component will be rolling updated after TiDB Operator is upgraded to v1.5.0 due to #5075.

11.2.6.2 New features

- Add the BR Federation Manager component to orchestrate `Backup` and `Restore` custom resources (CR) across multiple Kubernetes clusters (#4996, [@csuzhangxc](https://github.com/c
- Support using the `VolumeBackup` CR to back up a TiDB cluster deployed across multiple Kubernetes clusters based on EBS snapshots (#5013, [@WangLe1321](https://github.com/WangLe1321))

- Support using the `VolumeRestore` CR to restore a TiDB cluster deployed across multiple Kubernetes clusters based on EBS snapshots (#5039, [@WangLe1321](https://github.com/WangLe1321))
- Support using the `VolumeBackupSchedule` CR to automatically back up a TiDB cluster deployed across multiple Kubernetes clusters based on EBS snapshots (#5036, [@BornChanger](https://github.com/BornChanger))
- Support backing up CRs related to `TidbCluster` when backing up a TiDB cluster deployed across multiple Kubernetes based on EBS snapshots (#5207, [@WangLe1321](https://github.com/WangLe1321))

11.2.6.3 Improvements

- Add the `startUpScriptVersion` field for DM master to specify the version of the startup script (#4971, [@hanlins](https://github.com/hanlins))
- Support `spec.preferIPv6` for DmCluster, TidbDashboard, TidbMonitor, and TidbNGMonitoring (#4977, [@KanShiori](https://github.com/KanShiori))
- Support setting expiration time for TiKV leader eviction and PD leader transfer (#4997, [@Tema](https://github.com/Tema))
- Support setting toleration for `TidbInitializer` (#5047, [@csuzhangxc](https://github.com/csuzhan
- Support configuring the timeout for PD start (#5071, [@oliviachenairbnb](https://github.com/oliviac
- Skip evicting leaders for TiKV when changing PVC size to avoid leader eviction blocked caused by low disk space (#5101, [@csuzhangxc](https://github.com/csuzhangxc))
- Support updating annotations and labels in services for PD, TiKV, TiFlash, TiProxy, DM-master, and DM-worker (#4973, [@wxiaomou](https://github.com/wxiaomou))
- Enable volume resizing by default for PV expansion (#5167, [@liubog2008](https://github.com/liubo

11.2.6.4 Bug fixes

- Fix the quorum loss issue during TiKV upgrade due to some TiKV stores going down (#4979, [@Tema](https://github.com/Tema))
- Fix the quorum loss issue during PD upgrade due to some members going down (#4995, [@Tema](https://github.com/Tema))
- Fix the issue that TiDB Operator panics when no Kubernetes cluster-level permission is configured (#5058, [@liubog2008](https://github.com/liubog2008))
- Fix the issue that TiDB Operator might panic when `AdditionalVolumeMounts` is set for the `TidbCluster` CR (#5058, [@liubog2008](https://github.com/liubog2008))
- Fix the issue that `baseImage` for the `TidbDashboard` CR is parsed incorrectly when custom image registry is used (#5014, [@linkinghack](https://github.com/linkinghack))

11.2.7 TiDB Operator 1.5.0-beta.1 Release Notes

Release date: April 11, 2023

TiDB Operator version: 1.5.0-beta.1

11.2.7.1 New Feature

- Support using the `tidb.pingcap.com/pd-transfer-leader` annotation to restart PD Pod gracefully (#4896, [@luohao](https://github.com/luohao))
- Support using the `tidb.pingcap.com/tidb-graceful-shutdown` annotation to restart TiDB Pod gracefully (#4948, [@wxiaomou](https://github.com/wxiaomou))
- Support managing TiCDC with Advanced StatefulSet (#4881, [@charleszheng44](https://github.com
- Support managing TiProxy with Advanced StatefulSet (#4917, [@xhebox](https://github.com/xhebo
- Add a new field `bootstrapSQLConfigMapName` in the TiDB spec to specify an initialization SQL file on TiDB's first bootstrap (#4862, [@fgksgf](https://github.com/fgksgf))
- Allow users to define a strategy to restart failed backup jobs, enhancing backup stability (#4883, [@WizardXiao](https://github.com/WizardXiao)) (#4925, [@WizardXiao](https://github.com/WizardXiao))

11.2.7.2 Improvements

- Upgrade Kubernetes dependencies to v1.20 (#4954, [@KanShiori](https://github.com/KanShiori))
- Add the metrics for the reconciler and worker queue to improve observability (#4882, [@hanlins](https://github.com/hanlins))
- When rolling upgrade TiKV Pods, TiDB Operator will wait for the leader to transfer back to the upgraded Pod before upgrading the next TiKV pod. This helps to reduce performance degradation during rolling upgrade (#4863, [@KanShiori](https://github.com/KanShiori))
- Allow users to customize Prometheus scraping settings (#4846, [@coderplay](https://github.com/cod
- Support sharing some of TiDB's certificates with TiProxy (#4880, [@xhebox](https://github.com/xhe
- Configure the field `ipFamilyPolicy` as `PreferDualStack` for all components' Services when `spec.preferIPv6` is set to `true` (#4959, [@KanShiori](https://github.com/KanShiori))
- Add the metrics for counting errors about the reconciliation to improve observability (#4952, [@coderplay](https://github.com/coderplay))

11.2.7.3 Bug fixes

- Fix the issue that pprof endpoint is not reachable because the route conflicts with the metrics endpoint (#4874, [@hanlins](https://github.com/hanlins))

11.3 v1.4

11.3.1 TiDB Operator 1.4.7 Release Notes

Release date: July 26, 2023

TiDB Operator version: 1.4.7

11.3.1.1 Bug fixes

- Make `logBackupTemplate` optional in BackupSchedule CR (#5190, [@Ehco1996](https://github.com

11.3.2 TiDB Operator 1.4.6 Release Notes

Release date: July 19, 2023

TiDB Operator version: 1.4.6

11.3.2.1 Improvements

- Enable volume resizing by default (#5167, [@liubog2008](https://github.com/liubog2008))

11.3.2.2 Bug fixes

- Fix the issue of `Error loading shared library libresolv.so.2` when executing backup and restore with BR >=v6.6.0 (#4935, [@Ehco1996](https://github.com/Ehco1996))
- Fix the issue that graceful drain for TiCDC does not work if a non-SemVer image tag is used for TiCDC (#5173, [@csuzhangxc](https://github.com/csuzhangxc))

11.3.3 TiDB Operator 1.4.5 Release Notes

Release date: June 26, 2023

TiDB Operator version: 1.4.5

11.3.3.1 Improvements

- Add the metrics for fine-grained TidbCluster reconcile errors (#4952, [@coderplay](https://github.com/coderplay))
- Add the metrics for the reconciler and worker queue to improve observability (#4882, [@hanlins](https://github.com/hanlins))
- Introduce `startUpScriptVersion` field for DM master to specify the startup script version (#4971, [@hanlins](https://github.com/hanlins))
- Add support for rolling restart and scaling-in of TiCDC clusters when deploying TiCDC across multiple kubernetes clusters (#5040, [@charleszheng44](https://github.com/charleszheng44))

11.3.3.2 Bug fixes

- Suppress GC when scheduled backup is newly created (#4940, [@oliviachenairbnb](https://github.com
- Make `backupTemplate` optional in backup CR (#4956, [@Ehco1996](https://github.com/Ehco1996))
- Fix the issue that TiDB Operator panics if no Kubernetes cluster-level permission is configured (#5058, [@liubog2008](https://github.com/liubog2008))
- Fix the issue that TiDB Operator might panic if `AdditionalVolumeMounts` is set for TidbCluster (#5058, [@liubog2008](https://github.com/liubog2008))

11.3.4 TiDB Operator 1.4.4 Release Notes

Release date: March 13, 2023

TiDB Operator version: 1.4.4

11.3.4.1 New features

- Support using volume-snapshot backup and restore on a TiDB cluster with TiFlash (#4812, [@fengou1](https://github.com/fengou1))
- Show an accurate backup size in the volume-snapshot backup by calculating the backup size from snapshot storage usage (#4819, [@fengou1](https://github.com/fengou1))
- Support retries for snapshot backups in case of unexpected failures caused by Kubernetes job or pod issues (#4895, [@WizardXiao](https://github.com/WizardXiao))
- Support integrated management of log backup and snapshot backup in the `BackupSchedule` CR (#4904, [@WizardXiao](https://github.com/WizardXiao))

11.3.4.2 Bug fixes

- Fix the issue that sync fails when using a custom build of TiDB with image version which is not in semantic version format (#4920, [@sunxiaoguang](https://github.com/sunxiaoguang))
- Fix the issue that TiDB Operator cannot restore the volume-snapshot backup data for scale-in clusters by ensuring sequential PVC names (#4888, [@WangLe1321](https://github.com/WangLe1321))
- Fix the issue that volume-snapshot backup might lead to a panic when there is no block change between two snapshots (#4922, [@fengou1](https://github.com/fengou1))
- Fix the potential issue of volume-snapshot failure during the final stage of restore by adding a new check for encryption (#4914, [@fengou1](https://github.com/fengou1))

11.3.5 TiDB Operator 1.4.3 Release Notes

Release date: February 24, 2023

TiDB Operator version: 1.4.3

11.3.5.1 Bug fixes

- Fix the issue that the TiFlash metric server does not listen on correct IPv6 addresses when the `preferIPv6` configuration is enabled (#4850, [@KanShiori](https://github.com/KanShiori))
- Fix the issue that TiDB Operator keeps modifying EBS disks in AWS when the feature gate `VolumeModifying` is enabled and EBS parameters are missing in `StorageClass` (#4850, [@liubog2008](https://github.com/liubog2008))

11.3.6 TiDB Operator 1.4.2 Release Notes

Release date: February 3, 2023

TiDB Operator version: 1.4.2

11.3.6.1 Bug fix

- Fix the issue that TiFlash does not listen on IPv6 addresses when the `preferIPv6` configuration is enabled (#4850, [@KanShiori](https://github.com/KanShiori))

11.3.7 TiDB Operator 1.4.1 Release Notes

Release date: January 13, 2023

TiDB Operator version: 1.4.1

11.3.7.1 New features

- Support cleaning up failed instances of PD, TiKV and TiFlash components by removing the failed Pods and PVCs to handle unplanned failures of Kubernetes nodes in auto failure recovery feature (#4824, [@lalitkfk](https://github.com/lalitkfk))
  - To enable this feature, you need to configure `controllerManager.detectNodeFailure` in TiDB Operator Helm chart and configure the `app.kubernetes.io/auto-failure-recovery: "true"` annotation in the TidbCluster CR.

11.3.7.2 Improvements

- Support configuring `controllerManager.kubeClientQPS` and `controllerManager.kubeClientBurst` in TiDB Operator Helm chart to set QPS and Burst for the Kubernetes client in TiDB Controller Manager (#4830,[@Thearas](https://github.com/Thearas))

11.3.7.3 Bug fixes

- Fix the issue that TiDB Controller Manager panics without PV permission (#4837, [@csuzhangxc](https://github.com/csuzhangxc))

11.3.8 TiDB Operator 1.4.0 Release Notes

Release date: December 29, 2022

TiDB Operator version: 1.4.0

11.3.8.1 New features

- Support managing TiDB Dashboard in a separate `TidbDashboard` CRD (#4787, [@SabaPing](https://github.com/SabaPing))
- Support configuring Readiness Probe for TiKV and PD (#4763, [@mikechengwei](https://github.com/mikechengwei))
- Support backup and restore based on Amazon EBS volume-snapshot (#4698, [@gozssky](https://github.com/gozssky))

11.3.8.2 Improvements

- Support configuring `.spec.preferIPv6: true` for compatibility with IPv6 network environments (#4811, [@KanShiori](https://github.com/KanShiori))

11.3.8.3 Bug fixes

- Fix the issue that the backup based on EBS snapshot cannot be restored to a different namespace (#4795, [@fengou1](https://github.com/fengou1))
- Fix the issue that when the log backup stops occupying the Complete state, the caller mistakenly believes that the log backup CR has been completed, so that the log backup cannot be truncated (#4810, [@WizardXiao](https://github.com/WizardXiao))

11.3.9 TiDB Operator 1.4.0-beta.3 Release Notes

Release date: December 2, 2022

TiDB Operator version: 1.4.0-beta.3

11.3.9.1 New Feature

- Add experimental support for TiProxy (#4693), [@xhebox](https://github.com/xhebox)
- Snapshot backup and restore based on Amazon EBS becomes GA (#4784, [@fengou1](https://github.com/fengou1)). This feature has the following benefits:
  - Reduce the impact of backup on QPS to less than 5%
  - Back up and restore data in a short time. For example, finish backup within 1 hour and restore in 2 hours.

11.3.9.2 Bug fixes

- Fix typo in error messages (#4773, [@dveeden](https://github.com/dveeden))
- Fix the issue of volume-snapshot backup cleanup failure (#4783, [@fengou1](https://github.com/feng
- Fix backup failure when the TiDB cluster has massive TiKV nodes (40+) (#4784, [@fengou1](https://github.com/fengou1))

11.3.10 TiDB Operator 1.4.0-beta.2 Release Notes

Release date: November 11, 2022

TiDB Operator version: 1.4.0-beta.2

11.3.10.1 Bug fixes

- Fix the issue that `BackupSchedule` does not set prefix when using Azure Blob Storage (#4767, [@WizardXiao](https://github.com/WizardXiao))
- Upgrade AWS SDK to v1.44.72 to support the Asia Pacific (Jakarta) region (`ap-southeast-3`) in AWS (#4771, [@WizardXiao](https://github.com/WizardXiao))

11.3.11 TiDB Operator 1.4.0-beta.1 Release Notes

Release date: October 27, 2022

TiDB Operator version: 1.4.0-beta.1

11.3.11.1 New Feature

- TiDB Operator supports snapshot backup and restore based on Amazon EBS (experimental) (#4698, [@gozssky](https://github.com/gozssky)). This feature has the following benefits:
  - Reduce the impact of backup on QPS to less than 5%
  - Shorten the backup and restore time

11.3.11.2 Bug fixes

- Fix the issue that the log backup checkpoint ts will not be updated after TiDB Operator restarts (#4746, [@WizardXiao](https://github.com/WizardXiao))
- Fix the issue that log backup checkpoint ts will not be updated when TLS is enabled for the TiDB cluster (#4716, [@WizardXiao](https://github.com/WizardXiao))

11.3.12 TiDB Operator 1.4.0-alpha.1 Release Notes

Release date: September 26, 2022

TiDB Operator version: 1.4.0-alpha.1

11.3.12.1 Compatibility Change

- Due to changes in #4683, the volume modification feature is change to be disabled by default. If you want to resize PVCs of a component, you need to manually enable this feature.

11.3.12.2 Rolling Update Changes

- Due to changes in #4494, if you deploy TiCDC without setting the `log-file` configuration item, upgrading TiDB Operator to v1.4.0-alpha.1 causes TiCDC to rolling upgrade.

11.3.12.3 New Feature

- Support setting location labels for tidb-server automatically (#4663, [@glorv](https://github.com/glo
- Add new fields `spec.tikv.scalePolicy` and `spec.tiflash.scalePolicy` to scale multiple TiFlash and TiKV Pods simultaneously (#3881, [@lizhemingi](https://github.com/lizheming
- Add a new field `startScriptVersion` to choose start script for all components (#4505, [@KanShiori](https://github.com/KanShiori))
- Allow to use shortened label names to refer to some well-known Kubernetes labels in PD's location labels (#4688, [@glorv](https://github.com/glorv))
- Support point-in-time recovery from a snapshot backup and a log backup (#4648 #4682 #4694 #4695, [@WizardXiao](https://github.com/WizardXiao))
- Add a new feature gate `VolumeModifying` to enable the volume modification feature, and the feature is disable by default (#4683, [@liubog2008](https://github.com/liubog2008))

- Support changing the `StorageClass` of a `TidbCluster` component to modify the IOPS and throughput for AWS EBS volumes used by a cluster (#4683, [@liubog2008](https://github.com/liubog2008))
- Support setting the `--check-requirements` argument for BR (#4631, [@KanShiori](https://github.com/KanShiori))
- Support using the field `additionalContainers` to customize pod container configuration. If the container names in this field match with the ones generated by TiDB Operator, the container configurations will be merged into the default configuration. (#4530, [@mikechengwei](https://github.com/mikechengwei))

11.3.12.4 Improvements

- Optimize prometheus remoteWrite configuration for `TidbMonitor` (#4247, [@mikechengwei](https://github.com/mikechengwei))
- Add metrics port for TiFlash `Service` (#4470, [@mikechengwei](https://github.com/mikechengwei))
- Update the default value of the `log-file` configuration item for TiCDC to avoid overwriting `/dev/stderr` (#4494, [@KanShiori](https://github.com/KanShiori))

11.3.12.5 Bug fixes

- Fix the issue that cluster sync gets stuck caused by suspending a cluster when scaling (#4679, [@KanShiori](https://github.com/KanShiori))
- Fix the issue that TiDB Operator panics if PD spec is nil (#4679, [@KanShiori](https://github.com/mahjonp))

11.4 v1.3

11.4.1 TiDB Operator 1.3.10 Release Notes

Release date: February 24, 2023

TiDB Operator version: 1.3.10

11.4.1.1 Improvement

- Bump Go version to 1.19 to fix security vulnerabilities

11.4.2 TiDB Operator 1.3.9 Release Notes

Release date: October 10, 2022

TiDB Operator version: 1.3.9

11.4.2.1 Bug fix

- Fix the issue that PD upgrade will get stuck if the `acrossK8s` field is set but the `clusterDomain` field is not set (#4522, [@liubog2008](https://github.com/liubog2008))

11.4.3 TiDB Operator 1.3.8 Release Notes

Release date: September 13, 2022

TiDB Operator version: 1.3.8

11.4.3.1 New Feature

- Add some special annotations for TidbCluster to configure the minimum ready duration for TiDB, TiKV, and TiFlash. The minimum ready duration specifies the minimum number of seconds that a newly created Pod takes to be ready during a rolling upgrade (#4675, [@liubog2008](https://github.com/liubog2008))

11.4.3.2 Improvement

- Support graceful upgrade of a TiCDC Pod if the Pod version is v6.3.0 or later versions (#4697, [@overvenus](https://github.com/overvenus))

11.4.4 TiDB Operator 1.3.7 Release Notes

Release date: August 1, 2022

TiDB Operator version: 1.3.7

11.4.4.1 New Features

- Add the `suspendAction` field to suspend any component. If a component is suspended, the `StatefulSet` of the component will be deleted. (#4640, [@KanShiori](https://github.com/KanShiori))

11.4.4.2 Improvement

- After all PVCs are scaled up, recreate the `StatefulSet` of a component so that new PVCs use the desired storage size (#4620, [@KanShiori](https://github.com/KanShiori))
- To avoid the TiKV PVC scale-up process getting stuck, continue scale-up if a leader eviction times out (#4625, [@KanShiori](https://github.com/KanShiori))

11.4.4.3 Bug fixes

- Fix the issue that TiKV cannot be upgraded when using local storage (#4615, [@KanShiori](https://github.com/KanShiori))
- Fix the issue that backup files may leak after cleanup (#4617, [@KanShiori](https://github.com/KanS

11.4.5 TiDB Operator 1.3.6 Release Notes

Release date: July 5, 2022

TiDB Operator version: 1.3.6

11.4.5.1 Improvement

- To reduce the impact of PVC scale-up on cluster performance, scale up PVCs pod by pod and evict TiKV leader before resizing PVCs of TiKV (#4609, #4604, [@KanShiori](https://github.com/KanShiori))

11.4.6 TiDB Operator 1.3.5 Release Notes

Release date: June 29, 2022

TiDB Operator version: 1.3.5

11.4.6.1 New Features

- Support backing up data to and restoring data from Azure Blob Storage (#4534, [@xu21yingan](https://github.com/xu21yingan))

11.4.7 TiDB Operator 1.3.4 Release Notes

Release date: June 22, 2022

TiDB Operator version: 1.3.4

11.4.7.1 Improvement

- Add the `volumes` field in the status information of each component to show the volume status (#4540, [@KanShiori](https://github.com/KanShiori))

11.4.8 TiDB Operator 1.3.3 Release Notes

Release date: May 17, 2022

TiDB Operator version: 1.3.3

11.4.8.1 New Feature

- Add a new field `spec.tidb.service.port` to customize the tidb service port (#4512, [@KanShiori](https://github.com/KanShiori))

11.4.8.2 Bug fixes

- Fix the issue that evict leader scheduler may leak during cluster upgrade (#4522, [@KanShiori](https://github.com/KanShiori))
- Update the base image of `tidb-backup-manager` to fix incompatibility with ARM architecture (#4490, [@better0332](https://github.com/better0332))
- Fix the issue that TiDB Operator may panic when tidb Service does not have any Endpoints (#4500, [@mikechengwei](https://github.com/mikechengwei))
- Fix the issue that Labels and Annotations of the component Pods may be lost after TiDB Operator fails to access the Kubernetes server and retries (#4498, [@duduainankai](https://github.com/duduainankai))

11.4.9 TiDB Operator 1.3.2 Release Notes

Release date: March 18, 2022

TiDB Operator version: 1.3.2

11.4.9.1 Improvements

- Support for TiDB to run on Istio-enabled kubernetes clusters (#4445, [@rahilsh](https://github.com/
- Support multi-arch docker image (#4469, [@better0332](https://github.com/better0332))

11.4.10 TiDB Operator 1.3.1 Release Notes

Release date: February 24, 2022

TiDB Operator version: 1.3.1

11.4.10.1 Compatibility Change

- Due to the issues in #4434 and #4435, if you have deployed TiFlash v5.4.0 or later versions when using TiDB Operator v1.3.0 or v1.3.0-beta.1, you must upgrade TiDB Operator by taking the following steps to **avoid TiFlash losing metadata**.

1. In TidbCluster spec, if the `storage.rafe.dir` and `raft.kvstore_path` fields in TiFlash's config `spec.tiflash.config.config` are not explicitly configured, you need to add the `storage.raft.dir` field. If `storage.main.dir` is not explicitly configured, you need to add the field.

```
spec:
  # ...
  tiflash:
    config:
      config: |
        # ...
        [storage]
          [storage.main]
            dir = ["/data0/db"]
          [storage.raft]
            dir = ["/data0/db/kvstore/"]
```

2. Upgrade TiDB Operator.

11.4.10.2 New Feature

- Add a new field `spec.dnsPolicy` to support configuring `DNSPolicy` for Pods (#4420, [@handlerww](https://github.com/handlerww))

11.4.10.3 Improvement

- `tidb-lightning` Helm chart uses `local` backend as the default backend (#4426, [@KanShiori](https://github.com/KanShiori))

11.4.10.4 Bug fixes

- Fix the issue that if the `raft.kvstore_path` field or the `storage.raft.dir` field is not set in TiFlash's config, TiFlash will lose metadata after upgrading TiFlash to v5.4.0 or later versions when using TiDB Operator v1.3.0 or v1.3.0-beta.1 (#4430, [@KanShiori](https://github.com/KanShiori))
- Fix the issue that TiFlash v5.4.0 or later versions does not work if the `tmp_path` field is not set in TiFlash's config when using TiDB Operator v1.3.0 or v1.3.0-beta.1 (#4430, [@KanShiori](https://github.com/KanShiori))
- Fix the issue that TiDB cluster's PD components failed to start due to discovery service errors (#4440, [@liubog2008](https://github.com/liubog2008))

11.4.11 TiDB Operator 1.3.0 Release Notes

Release date: February 15, 2022

TiDB Operator version: 1.3.0

11.4.11.1 Compatibility Change

- Due to changes in #4400, if a TiDB cluster is deployed across multiple Kubernetes clusters by TiDB Operator (<= v1.3.0-beta.1), upgrading TiDB Operator to v1.3.0 directly will cause failed rolling upgrade. You have to upgrade TiDB Operator by the following steps:

    1. Update CRD.
    2. Add a new `spec.acrossK8s` field in TidbCluster spec and set it to `true`.
    3. Upgrade TiDB Operator.

- Due to the issue in #4434, if you upgrade TiFlash to v5.4.0 or later when using v1.3.0 TiDB Operator, TiFlash might lose metadata and not work. If TiFlash is deployed in your cluster, it is recommended that you upgrade TiDB Operator to v1.3.1 or later versions before upgrading TiFlash.

- Due to the issue in #4435, when using TiDB Operator v1.3.0, you must set the `tmp_path` field in TiFlash's config to use TiFlash v5.4.0 or later versions. It is recommended that you upgrade TiDB Operator to v1.3.1 or later versions before you deploy TiFlash.

11.4.11.2 New Features

- Add a new field `spec.tidb.tlsClient.skipInternalClientCA` to skip server certificate verification when internal components access TiDB (#4388, [@just1900](https://github.com/just
- Support configuring DNS config for Pods of all components (#4394, [@handlerww](https://github.com/handlerww))
- Add a new field `spec.tidb.initializer.createPassword` to support setting a random password for TiDB when deploying a new cluster (#4328, [@mikechengwei](https://github.com/mikechengwei))
- Add a new field `failover.recoverByUID` to support one-time recover for TiKV/TiFlash/DM Worker (#4373, [@better0332](https://github.com/better0332))
- Add a new field `spec.pd.startUpScriptVersion` to use the `dig` command instead of `nslookup` to lookup domain in the startup script of PD (#4379, [@july2993](https://github.com/july2993))

11.4.11.3 Improvements

- Pre-check whether `VolumeMount` exists when the StatefuleSet of components is deployed or updated to avoid failed rolling upgrade (#4369, [@july2993](https://github.com/july2993))
- Enhance the feature of deploying a TiDB cluster across Kubernetes clusters:
  - Add a new field `spec.acrossK8s` to indicate deploying a TiDB cluster across Kubernetes clusters (#4400, [@KanShiori](https://github.com/KanShiori))
  - Support deploying heterogeneous TiDB cluster across Kubernetes clusters (#4387, [@KanShiori](https://github.com/KanShiori))
  - The field `spec.clusterDomain` is not required when TiDB cluster is deployed across Kubernetes clusters. The field is only used for addresses accessed between components (#4408, [@KanShiori](https://github.com/KanShiori))
  - Fix the issue that in cross-Kubernetes deployment, Pump becomes abnormal when all PDs of one Kubernetes cluster are down (#4377, [@just1900](https://github.com/just1900))

11.4.11.4 Bug fixes

- Fix the issue that tidb scheduler cannot be deployed on Kubernetes v1.23 or later versions (#4386, [@just1900](https://github.com/just1900))

11.4.12 TiDB Operator 1.3.0-beta.1 Release Notes

Release date: January 12, 2022

TiDB Operator version: 1.3.0-beta.1

11.4.12.1 Compatibility Change

- Due to changes in #4209, if Webhook is deployed, and `ValidatingWebhook` and `MutatingWebhook` of Pods are enabled with TiDB Operator v1.2 or earlier versions, upgrading TiDB Operator to v1.3.0-beta.1 will cause `ValidatingWebhook` and `MutatingWebhook` to be deleted. But this has no impact on TiDB cluster management.
- Due to changes in #4151, if you deploy v1 CRD, TiDB Operator >= v1.3.0-beta.1 sets the default `baseImage` field of all components. If you set the component image using the `image` field instead of the `baseImage` field, upgrading TiDB Operator to v1.3.0-beta.1 will change the image in use, cause the TiDB cluster to rolling update or even fail to run. To avoid such situations, you must upgrade TiDB Operator by the following steps:

  1. Use the `baseImage` and `version` fields to replace the `image` field. For details, refer to Configure TiDB deployment.
  2. Upgrade TiDB Operator.

- Due to the issue in #4434, if you upgrade TiFlash to v5.4.0 or later when using v1.3.0-beta.1 TiDB Operator, TiFlash might lose metadata and not work. If TiFlash is deployed in your cluster, it is recommended that you upgrade TiDB Operator to v1.3.1 or later versions before upgrading TiFlash.
- Due to the issue in #4435, when using TiDB Operator v1.3.0-beta.1, you must set the `tmp_path` field in TiFlash's config to use TiFlash v5.4.0 or later versions. It is recommended that you upgrade TiDB Operator to v1.3.1 or later versions before you deploy TiFlash.

11.4.12.2 Rolling Update Changes

- Due to changes in #4358, if the TiDB cluster (>= v5.4) is deployed by TiDB Operator v1.2, upgrading TiDB Operator to v1.3.0-beta.1 causes TiFlash to rolling upgrade. It is recommended to upgrade TiDB Operator to v1.3 before upgrading the TiDB cluster to v5.4.0 or later versions.
- Due to changes in #4169 , for TiDB clusters >= v5.0, if `spec.tikv.separateRocksDBLog` ↪ `: true` or `spec.tikv.separateRaftLog: true` is configured, upgrading TiDB Operator to v1.3.0-beta.1 causes TiKV to rolling upgrade.
- Due to changes in #4198, upgrading TiDB Operator causes the recreation of TidbMonitor Pod.

11.4.12.3 New Features

- Support configuring the resource usage for the init container of TiFlash (#4304, [@KanShiori](https://github.com/KanShiori))
- Support enabling continuous profiling for the TiDB cluster (#4287, [@KanShiori](https://github.com/KanShiori))
- Support gracefully restarting TiKV through annotations (#4279, [@july2993](https://github.com/jul
- Support `PodSecurityContext` and other configurations for Discovery (#4259, [@csuzhangxc](https://github.com/csuzhangxc), #4208, [@KanShiori](https://github.com/KanShio
- Support configuring `PodManagementPolicy` in TidbCluster CR (#4211, [@mianhk](https://github.com/mianhk))
- Support configuring Prometheus shards in TidbMonitor CR (#4198, [@mikechengwei](https://github.com/mikechengwei))
- Support deploying TiDB Operator on Kubernetes v1.22 or later versions (#4195, #4202, [@KanShiori](https://github.com/KanShiori))
- Generate v1 CRD to support deploying on Kubernetes v1.22 or later versions (#4151, [@KanShiori](https://github.com/KanShiori))

11.4.12.4 Improvements

- Remove and change some default configurations for TiFlash due to configuration changes in TiFlash v5.4.0. If the TiDB cluster (>= v5.4) is deployed by TiDB Operator v1.2, upgrading TiDB Operator to v1.3.0-beta.1 causes TiFlash to rolling upgrade. (#4358, [@KanShiori](https://github.com/KanShiori))
- Improve advanced deployment example of TidbMonitor. (#4242, [@mianhk](https://github.com/mia
- Optimize the user experience of heterogenous clusters by displaying the metrics for one TiDB cluster and its heterogeneous clusters in the same dashboards. (#4210, [@mikechengwei](https://github.com/mikechengwei))
- Use `secretRef` to obtain Grafana password in TidbMonitor to avoid using plaintext password. (#4135, [@mianhk](https://github.com/mianhk))
- Optimize the upgrade process for PD and TiKV components with fewer than two replicas, and force the upgrade of PD and TiKV components by default to avoid the upgrade process from taking too long (#4107, #4091, [@mianhk](https://github.com/mianhk))
- Update Grafana images in examples to 7.5.11 to enhance security (#4212, [@makocchi-git](https://github.com/makocchi-git))
- Deprecate Pod validating and mutating webhook (#4209, [@mianhk](https://github.com/mianhk))
- Support configuring the number of tidb-controller-manager workers in Helm chart (#4111, [@mianhk](https://github.com/mianhk))

11.5 v1.2

11.5.1 TiDB Operator 1.2.7 Release Notes

Release date: February 17, 2022

TiDB Operator version: 1.2.7

11.5.1.1 New Features

- Add a new field `spec.pd.startUpScriptVersion` to use the `dig` command instead of `nslookup` to lookup domain in the startup script of PD (#4379, [@july2993](https://github.com/july2993))

11.5.1.2 Improvements

- Pre-check whether `VolumeMount` exists when the StatefuleSet of components is deployed or updated to avoid failed rolling upgrade (#4369, [@july2993](https://github.com/july2993))

11.5.2 TiDB Operator 1.2.6 Release Notes

Release date: January 4, 2022

TiDB Operator version: 1.2.6

11.5.2.1 Improvements

- Refine retry logic when updating the status of the Backup and Restore CR (#4337, [@just1900](https://github.com/just1900))

11.5.3 TiDB Operator 1.2.5 Release Notes

Release date: December 27, 2021

TiDB Operator version: 1.2.5

11.5.3.1 Improvements

- Support configuring all fields in `ComponentSpec` for DM to control component behavior more finely (#4313, [@csuzhangxc](https://github.com/csuzhangxc))
- Support configuring init container `resources` for TiFlash (#4304, [@KanShiori](https://github.com/KanShiori))
- Support configuring the `ssl-ca` parameter for TiDB via `TiDBTLSClient` to disable client authentication (#4270, [@just1900](https://github.com/just1900))
- Add a `ready` field to TiCDC captures to show its readiness status (#4273, [@KanShiori](https://github.com/KanShiori))

11.5.3.2 Bug fixes

- Fix the issue that PD, TiKV, and TiDB cannot roll update after the component startup script is updated (#4248, [@KanShiori](https://github.com/KanShiori))
- Fix the issue that the TidbCluster spec is updated automatically after TLS is enabled (#4306, [@KanShiori](https://github.com/KanShiori))
- Fix a potential goroutine leak when TiDB Operator checks the Region leader count of TiKV (#4291, [@DanielZhangQD](https://github.com/DanielZhangQD))
- Fix some high-level security issues (#4240, [@KanShiori](https://github.com/KanShiori))

11.5.4 TiDB Operator 1.2.4 Release Notes

Release date: October 21, 2021

TiDB Operator version: 1.2.4

11.5.4.1 Rolling update changes

- Upgrading TiDB Operator will cause the recreation of the TiDBMonitor Pod due to #4180

11.5.4.2 New features

- TidbMonitor supports customizing prometheus rules and reloading configurations dynamically (#4180, [@mikechengwei](https://github.com/mikechengwei))
- TidbMonitor supports the `enableRules` field. When AlertManager is not configured, you can configure this field to `true` to add Prometheus rules (#4115, [@mikechengwei](https://github.com/mikechengwei))

11.5.4.3 Improvements

- Optimize `TiFlash` rolling upgrade process (#4193, [@KanShiori](https://github.com/KanShiori))
- Support deleting backup data in batches (#4095, [@KanShiori](https://github.com/KanShiori))

11.5.4.4 Bug fixes

- Fix the security vulnerabilities in the `tidb-backup-manager` and `tidb-operator` images (#4217, [@KanShiori](https://github.com/KanShiori))
- Fix the issue that some backup data might retain if the `Backup` CR is deleted when the `Backup` job is running (#4133, [@KanShiori](https://github.com/KanShiori))

11.5.5 TiDB Operator 1.2.3 Release Notes

Release date: September 7, 2021

TiDB Operator version: 1.2.3

11.5.5.1 Bug fixes

- Fix the TiFlash Pod rolling recreation issue after TiDB Operator is upgraded to v1.2.2 (#4181, [@KanShiori](https://github.com/KanShiori))

11.5.6 TiDB Operator 1.2.2 Release Notes

Release date: September 3, 2021

TiDB Operator version: 1.2.2

11.5.6.1 Rolling update changes

- Upgrading TiDB Operator will cause the recreation of the TiDBMonitor Pod due to #4158
- Upgrading TiDB Operator will cause the recreation of the TiFlash Pod due to #4152

11.5.6.2 New features

- TiDBMonitor supports reloading configurations dynamically (#4158, [@mikechengwei](https://github.com/mikechengwei))

11.5.6.3 Bug fixes

- Fix upgrade failures of TiCDC from an earlier version to v5.2.0 (#4171, [@KanShiori](https://github.com/KanShiori))

11.5.7 TiDB Operator 1.2.1 Release Notes

Release date: August 18, 2021

TiDB Operator version: 1.2.1

11.5.7.1 Rolling update changes

- If `hostNetwork` is enabled for TiCDC, upgrading TiDB Operator will cause the recreation of the TiCDC Pod due to #4141

11.5.7.2 Improvements

- Support configuring `hostNetwork` for all components in TidbCluster so that all components can use host network (#4141, [@DanielZhangQD](https://github.com/DanielZhangQD))

11.5.8 TiDB Operator 1.2.0 Release Notes

Release date: July 29, 2021

TiDB Operator version: 1.2.0

11.5.8.1 Rolling update changes

- Upgrading TiDB Operator will cause the recreation of the TidbMonitor Pod due to #4085

11.5.8.2 New features

- Support setting Prometheus `retentionTime`, which is more fine-grained than `reserveDays`, and only `retentionTime` takes effect if both are configured (#4085, [@better0332](https://github.com/better0332))
- Support setting `priorityClassName` in the `Backup` CR to specify the priority of the backup job (#4078, [@mikechengwei](https://github.com/mikechengwei))

11.5.8.3 Improvements

- Changes the default Region leader eviction timeout of TiKV to 1500 minutes. The change prevents the Pod from being deleted when the Region leaders are not transferred completely to the other stores, which will cause data corruption (#4071, [@KanShiori](https://github.com/KanShiori))

11.5.8.4 Bug fixes

- Fix the issue that the URL in `Prometheus.RemoteWrite` may be parsed incorrectly in `TiDBMonitor` (#4087, [@better0332](https://github.com/better0332))

11.5.9 TiDB Operator 1.2.0-rc.2 Release Notes

Release date: July 2, 2021

TiDB Operator version: 1.2.0-rc.2

11.5.9.1 New features

- Support passing raw TOML config for TiCDC (#4010, [@july2993](https://github.com/july2993))
- Support setting `StorageVolumes`, `AdditionalVolumes`, and `AdditionalVolumeMounts` for TiCDC (#4004, [@csuzhangxc](https://github.com/csuzhangxc))
- Support setting custom `labels` and `annotations` for Discovery, TidbMonitor, and TidbInitializer (#4029, [@csuzhangxc](https://github.com/csuzhangxc))
- Support modifying Grafana dashboard (#4035, [@mikechengwei](https://github.com/mikechengwei)

11.5.9.2 Improvements

- 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
- Add the liveness and readiness probes for TiDB Operator to check the TiDB Operator status (#4002, [@mikechengwei](https://github.com/mikechengwei))

11.5.9.3 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 the status of TiDB service and TidbCluster are updated continuously even when no change is made to the TidbCluster spec (#4008, [@july2993](https://github.com/july2993))

11.5.10 TiDB Operator 1.2.0-rc.1 Release Notes

Release date: May 28, 2021

TiDB Operator version: 1.2.0-rc.1

11.5.10.1 Rolling update changes

- Upgrading TiDB Operator will cause the recreation of the Pump Pod due to #3973

11.5.10.2 New features

- Support customized labels for TidbCluster Pods and services (#3892, [@SabaP-ing](https://github.com/SabaPing) [@july2993](https://github.com/july2993))
- Support full lifecycle management for Pump (#3973, [@july2993](https://github.com/july2993))

11.5.10.3 Improvements

- Mask the backup password in logging (#3979, [@dveeden](https://github.com/dveeden))
- Add an additional volumeMounts field for Grafana (#3960, [@mikechengwei](https://github.com/mik
- Add several useful additional printout columns for TidbMonitor (#3958, [@mikecheng-wei](https://github.com/mikechengwei))
- TidbMonitor supports writing monitor configuration to PD etcd directly (#3924, [@mikechengwei](https://github.com/mikechengwei))

11.5.10.4 Bug fixes

- Fix the issue that TidbMonitor may not work for DmCluster with TLS enabled (#3991, [@csuzhangxc](https://github.com/csuzhangxc))
- Fix the wrong count of PD members when scaling out PD (#3940, [@cvvz](https://github.com/cvvz))
- Fix the issue that DM-master might fail to restart (#3972, [@csuzhangxc](https://github.com/csuzha
- Fix the issue that rolling update might happen after changing `configUpdateStrategy` from `InPlace` to `RollingUpdate` (#3970, [@cvvz](https://github.com/cvvz))
- Fix the issue that backup using Dumpling might fail (#3986, [@liubog2008](https://github.com/liubo

11.5.11 TiDB Operator 1.2.0-beta.2 Release Notes

Release date: April 29, 2021

TiDB Operator version: 1.2.0-beta.2

11.5.11.1 Rolling update changes

- Upgrading TiDB Operator will cause the recreation of the TidbMonitor Pod due to #3943
- Upgrading TiDB Operator will cause the recreation of the DM-master Pod due to #3914

11.5.11.2 New features

- TidbMonitor supports monitoring multiple TidbClusters with TLS enabled (#3867, [@mikechengwei](https://github.com/mikechengwei))
- Support configuring `podSecurityContext` for all TiDB components (#3909, [@liubog2008](https://github.com/liubog2008))
- Support configuring `topologySpreadConstraints` for all TiDB components (#3937, [@liubog2008](https://github.com/liubog2008))
- Support deploying a DmCluster in a different namespace than a TidbCluster (#3914, [@csuzhangxc](https://github.com/csuzhangxc))
- Support installing TiDB Operator with only namespace-scoped permissions (#3896, [@csuzhangxc](https://github.com/csuzhangxc))

11.5.11.3 Improvements

- Add the readiness probe for the TidbMonitor Pod (#3943, [@mikechengwei](https://github.com/mike
- Optimize TidbMonitor for DmCluster with TLS enabled (#3942, [@mikechengwei](https://github.com/mikechengwei))
- TidbMonitor supports not generating Prometheus alert rules (#3932, [@mikechengwei](https://github.com/mikechengwei))

11.5.11.4 Bug fixes

- Fix the issue that TiDB instances are kept in TiDB Dashboard after being scaled in (#3929, [@july2993](https://github.com/july2993))
- Fix the useless sync of TidbCluster CR caused by the update of `lastHeartbeatTime` in `status.tikv.stores` (#3886, [@songjiansuper](https://github.com/songjiansuper))

11.5.12 TiDB Operator 1.2.0-beta.1 Release Notes

Release date: April 7, 2021

TiDB Operator version: 1.2.0-beta.1

11.5.12.1 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).

11.5.12.2 Rolling Update Changes

- Upgrading TiDB Operator will cause the recreation of the TidbMonitor Pod due to #3785

11.5.12.3 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))
- Support affinity and tolerations in backup/restore CR (#3835, [@dragonly](https://github.com/drago
- The resources in the tidb-operator chart use the new service account when `appendReleaseSuffix` is set to `true` (#3819, [@DanielZhangQD](https://github.com/DanielZhangQ
- Support configuring durations for leader election (#3794, [@july2993](https://github.com/july2993))
- Add the `tidb_cluster` label for the scrape jobs in TidbMonitor to support monitoring multiple clusters (#3750, [@mikechengwei](https://github.com/mikechengwei))
- Support setting customized store labels according to the node labels (#3784, [@L3T](https://github.com/L3T))
- Support customizing the storage config for TiDB slow log (#3731, [@BinChenn](https://github.com/B
- 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))

11.5.12.4 Improvements

- Add retry for DNS lookup failure exception in TiDBInitializer (#3884, [@handlerww](https://github.com/handlerww))

- Optimize thanos example yaml files (#3726, [@mikechengwei](https://github.com/mikechengwei))
- Delete the evict leader scheduler after TiKV Pod is recreated during the rolling update (#3724, [@handlerww](https://github.com/handlerww))
- Support multiple PVCs for PD during scaling and failover (#3820, [@dragonly](https://github.com/dragonly))
- Support multiple PVCs for TiKV during scaling (#3816, [@dragonly](https://github.com/dragonly))
- Support PVC resizing for TiDB (#3891, [@dragonly](https://github.com/dragonly))
- 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))

11.5.12.5 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 issue that some unrecognized environment variables are included in the external labels of the TidbMonitor (#3785, [@mikechengwei](https://github.com/mikechengwei))
- 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 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 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 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))

11.5.13 TiDB Operator 1.2.0-alpha.1 Release Notes

Release date: January 15, 2021

TiDB Operator version: 1.2.0-alpha.1

11.5.13.1 Rolling Update Changes

- Due to #3440, the Pod of TidbMonitor will be deleted and recreated after TiDB Operator is upgraded to v1.2.0-alpha.1.

11.5.13.2 New Features

- Deploy one TiDB cluster across multiple Kubernetes clusters ([@L3T](https://github.com/L3T), [@handlerww](https://github.com/handlerww))
- Support DM 2.0 in TiDB Operator ([@lichunzhu](https://github.com/lichunzhu), [@BinChenn](https://github.com/BinChenn))
- Auto-Scaling with PD API ([@howardlau1999](https://github.com/howardlau1999))
- Canary Upgrade of TiDB Operator (#3548, [@shonge](https://github.com/shonge), #3554, [@cvvz](https://github.com/cvvz))

11.5.13.3 Improvements

- Add local backend support for the TiDB Lightning chart (#3644, [@csuzhangxc](https://github.com/
- Add TLS support for the TiDB Lightning chart and TiKV Importer chart (#3598, [@csuzhangxc](https://github.com/csuzhangxc))
- Support persisting checkpoint for TiDB Lightning helm chart (#3653, [@csuzhangxc](https://github.c
- Support Thanos sidecar for monitoring multiple clusters (#3579, [@mikechengwei](https://github.com/mikechengwei))
- Migrate from Deployment to StatefulSet for TidbMonitor (#3440, [@mikechengwei](https://github.com/mikechengwei))

11.5.13.4 Other Notable Changes

- Optimize rate limiter intervals (#3700, [@dragonly](https://github.com/dragonly))
- Change the directory to save the customized alert rules in TidbMonitor from `tidb:${tidb_image_version}` to `tidb:${initializer_image_version}`. Please note that if the `spec.initializer.version` in the TidbMonitor does not match with the TiDB version in the TidbCluster, upgrading TiDB Operator will cause the re-creation of the monitor Pod (#3684, [@BinChenn](https://github.com/BinChenn))

11.6 v1.1

11.6.1 TiDB Operator 1.1.15 Release Notes

Release date: February 17, 2022

TiDB Operator version: 1.1.15

11.6.1.1 Bug Fixes

- Fix a potential goroutine leak when TiDB Operator checks the Region leader count of TiKV (#4291, [@DanielZhangQD](https://github.com/DanielZhangQD))

11.6.2 TiDB Operator 1.1.14 Release Notes

Release date: October 21, 2021

TiDB Operator version: 1.1.14

11.6.2.1 Bug Fixes

- Fix the security vulnerabilities in the `tidb-backup-manager` and `tidb-operator` images (#4217, [@KanShiori](https://github.com/KanShiori))

11.6.3 TiDB Operator 1.1.13 Release Notes

Release date: July 2, 2021

TiDB Operator version: 1.1.13

11.6.3.1 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))

11.6.3.2 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))

11.6.4 TiDB Operator 1.1.12 Release Notes

Release date: April 15, 2021

TiDB Operator version: 1.1.12

11.6.4.1 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

11.6.4.2 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))

11.6.4.3 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))

11.6.5 TiDB Operator 1.1.11 Release Notes

Release date: February 26, 2021

TiDB Operator version: 1.1.11

11.6.5.1 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))

11.6.5.2 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))

11.6.6 TiDB Operator 1.1.10 Release Notes

Release date: January 28, 2021

TiDB Operator version: 1.1.10

11.6.6.1 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).

11.6.6.2 Rolling Update Changes

- Upgrading TiDB Operator will cause the recreation of the TidbMonitor Pod due to #3684

11.6.6.3 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

11.6.6.4 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))

11.6.6.5 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))

11.6.7 TiDB Operator 1.1.9 Release Notes

Release date: December 28, 2020

TiDB Operator version: 1.1.9

11.6.7.1 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)

11.6.7.2 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))

11.6.8 TiDB Operator 1.1.8 Release Notes

Release date: December 21, 2020

TiDB Operator version: 1.1.8

11.6.8.1 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))

11.6.8.2 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, for example, 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))

11.6.8.3 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

11.6.9 TiDB Operator 1.1.7 Release Notes

Release date: November 13, 2020

TiDB Operator version: 1.1.7

11.6.9.1 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`

11.6.9.2 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))

11.6.9.3 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))

11.6.9.4 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))

11.6.10 TiDB Operator 1.1.6 Release Notes

Release date: October 16, 2020

TiDB Operator version: 1.1.6

11.6.10.1 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`

11.6.10.2 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.

11.6.10.3 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))

11.6.10.4 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))

11.6.10.5 Bug Fixes

- Fix the problem that may bootstrap multiple PD clusters (#3365, [@lichunzhu](https://github.com/lichunzhu))

11.6.11 TiDB Operator 1.1.5 Release Notes

Release date: September 18, 2020

TiDB Operator version: 1.1.5

11.6.11.1 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))

11.6.11.2 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))

11.6.11.3 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))

11.6.11.4 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))

11.6.12 TiDB Operator 1.1.4 Release Notes

Release date: August 21, 2020

TiDB Operator version: 1.1.4

11.6.12.1 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/DanielZhang
- 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))

11.6.13 TiDB Operator 1.1.3 Release Notes

Release date: July 27, 2020

TiDB Operator version: 1.1.3

11.6.13.1 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))

11.6.13.2 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))

11.6.14 TiDB Operator 1.1.2 Release Notes

Release date: July 1, 2020

TiDB Operator version: 1.1.2

11.6.14.1 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))

11.6.14.2 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))

11.6.15 TiDB Operator 1.1.1 Release Notes

Release date: June 19, 2020

TiDB Operator version: 1.1.1

11.6.15.1 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

11.6.16 TiDB Operator 1.1 GA Release Notes

Release date: May 28, 2020

TiDB Operator version: 1.1.0

11.6.16.1 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.

11.6.16.2 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`.

11.6.16.3 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))

11.6.17 TiDB Operator 1.1 RC.4 Release Notes

Release date: May 15, 2020

TiDB Operator version: 1.1.0-rc.4

11.6.17.1 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))

11.6.17.2 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 (for example, 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))

11.6.18 TiDB Operator 1.1 RC.3 Release Notes

Release date: April 30, 2020

TiDB Operator version: 1.1.0-rc.3

11.6.18.1 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))

11.6.19 TiDB Operator 1.1 RC.2 Release Notes

Release date: April 15, 2020

TiDB Operator version: 1.1.0-rc.2

11.6.19.1 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))

11.6.19.2 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

11.6.20 TiDB Operator 1.1 RC.1 Release Notes

Release date: April 1, 2020

TiDB Operator version: 1.1.0-rc.1

11.6.20.1 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))

11.6.20.2 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

11.6.21 TiDB Operator 1.1 Beta.2 Release Notes

Release date: February 26, 2020

TiDB Operator version: 1.1.0-beta.2

11.6.21.1 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))

11.6.21.2 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

11.6.22 TiDB Operator 1.1 Beta.1 Release Notes

Release date: January 8, 2020

TiDB Operator version: 1.1.0-beta.1

11.6.22.1 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`.

11.6.22.2 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` outputs 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 configuration (#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/DanielZhangQD
- 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 podSecurityContext 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))

11.7 v1.0

11.7.1 TiDB Operator 1.0.7 Release Notes

Release date: June 16, 2020

TiDB Operator version: 1.0.7

11.7.1.1 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)

11.7.2 TiDB Operator 1.0.6 Release Notes

Release date: December 27, 2019

TiDB Operator version: 1.0.6

11.7.2.1 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
  ...
```

11.7.2.1.1 Monitor

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

11.7.2.1.2 TiDB Scheduler

- Refine scheduler error messages (#1373)

11.7.2.1.3 Compatibility

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

11.7.2.1.4 TiKV Importer

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

11.7.2.1.5 E2E

- Ensure pods unaffected when upgrading (#955)

11.7.2.1.6 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)

11.7.3 TiDB Operator 1.0.5 Release Notes

Release date: December 11, 2019

TiDB Operator version: 1.0.5

11.7.3.1 v1.0.5 What's New

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

11.7.3.1.1 Scheduled Backup

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

11.7.3.1.2 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)

11.7.3.1.3 Compatibility

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

11.7.3.1.4 E2E

- Fix e2e nil point dereference (#1221)

11.7.4 TiDB Operator 1.0.4 Release Notes

Release date: November 23, 2019

TiDB Operator version: 1.0.4

11.7.4.1 v1.0.4 What's New

11.7.4.1.1 Action Required

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

11.7.4.1.2 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 on Kubernetes. Check out the document for detailed user guide.

Another new Helm chart `tidb-drainer` brings multiple drainers support for TiDB Binlog on Kubernetes. Check out the document for detailed user guide.

11.7.4.1.3 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)

11.7.4.2 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 `podSecurityContext` to nil by default in favor of backward compatibility (#1184)

11.7.4.3 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)

11.7.5 TiDB Operator 1.0.3 Release Notes

Release date: November 13, 2019

TiDB Operator version: 1.0.3

11.7.5.1 v1.0.3 What's New

11.7.5.1.1 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`.

11.7.5.1.2 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

11.7.5.2 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)

11.7.6 TiDB Operator 1.0.2 Release Notes

Release date: November 1, 2019

TiDB Operator version: 1.0.2

11.7.6.1 v1.0.2 What's New

11.7.6.1.1 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.

11.7.6.1.2 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

11.7.6.1.3 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

11.7.6.2 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)

11.7.7 TiDB Operator 1.0.1 Release Notes

Release date: September 17, 2019

TiDB Operator version: 1.0.1

11.7.7.1 v1.0.1 What's New

11.7.7.1.1 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.

11.7.7.1.2 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`

11.7.7.1.3 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

11.7.7.2 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)

11.7.8 TiDB Operator 1.0 GA Release Notes

Release date: July 30, 2019

TiDB Operator version: 1.0.0

11.7.8.1 v1.0.0 What's New

11.7.8.1.1 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.

11.7.8.1.2 Stability Test Cases Added

- Stop all etcds and kubelets

11.7.8.1.3 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

11.7.8.1.4 Bug Fixes

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

11.7.8.2 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)

11.7.9 TiDB Operator 1.0 RC.1 Release Notes

Release date: July 12, 2019

TiDB Operator version: 1.0.0-rc.1

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

11.7.9.1.1 Stability test cases added

- Stop kube-proxy
- Upgrade tidb-operator

11.7.9.1.2 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.storeLabels` setting
- On GKE one can use COS for TiKV nodes with small data for faster startup
- Support force upgrade when PD cluster is unavailable.

11.7.9.1.3 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

11.7.9.2 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)

11.7.10 TiDB Operator 1.0 Beta.3 Release Notes

Release date: June 6, 2019

TiDB Operator version: 1.0.0-beta.3

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

11.7.10.1.1 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.

11.7.10.1.2 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

11.7.10.1.3 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, for example, 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

11.7.10.1.4 Documentation Improvement

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

11.7.10.1.5 Pass User Acceptance Tests

11.7.10.1.6 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

11.7.10.2 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)

11.7.11 TiDB Operator 1.0 Beta.2 Release Notes

Release date: May 10, 2019

TiDB Operator version: 1.0.0-beta.2

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

11.7.11.1.1 Stability has been greatly enhanced

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

11.7.11.1.2 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

11.7.11.1.3 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

11.7.11.1.4 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

11.7.11.2 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)

11.7.12 TiDB Operator 1.0 Beta.1 P2 Release Notes

Release date: February 21, 2019

TiDB Operator version: 1.0.0-beta.1-p2

11.7.12.1 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)

11.7.13 TiDB Operator 1.0 Beta.1 P1 Release Notes

Release date: January 7, 2019

TiDB Operator version: 1.0.0-beta.1-p1

11.7.13.1 Bug Fixes

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

11.7.13.2 Docs

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

11.7.14 TiDB Operator 1.0 Beta.1 Release Notes

Release date: December 27, 2018

TiDB Operator version: 1.0.0-beta.1

11.7.14.1 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)

11.7.14.2 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)

11.7.15 TiDB Operator 1.0 Beta.0 Release Notes

Release date: November 26, 2018

TiDB Operator version: 1.0.0-beta.0

11.7.15.1 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)

11.8 v0

11.8.1 TiDB Operator 0.4 Release Notes

Release date: November 9, 2018

TiDB Operator version: 0.4.0

11.8.1.1 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)

11.8.1.2 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)

11.8.2 TiDB Operator 0.3.1 Release Notes

Release date: October 31, 2018

TiDB Operator version: 0.3.1

11.8.2.1 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)

11.8.2.2 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)

11.8.3 TiDB Operator 0.3.0 Release Notes

Release date: October 12, 2018

TiDB Operator version: 0.3.0

11.8.3.1 Notable Changes

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

11.8.4 TiDB Operator 0.2.1 Release Notes

Release date: September 20, 2018

TiDB Operator version: 0.2.1

11.8.4.1 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)

11.8.5 TiDB Operator 0.2.0 Release Notes

Release date: September 11, 2018

TiDB Operator version: 0.2.0

11.8.5.1 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

11.8.6 TiDB Operator 0.1.0 Release Notes

Release date: August 22, 2018

TiDB Operator version: 0.1.0

11.8.6.1 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
