TiDB on Kubernetes Documentation

PingCAP Inc.

20250718

Table of Contents

1 TiDB on Kubernetes Docs

2 Introduction

2.1 TiDB Operator Overview

TiDB Operator is an automated operating 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.

2.1.1 Compatibility between TiDB Operator and TiDB

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

TiDB versions	Compatible TiDB Operator versions
dev	dev
TiDB >= 8.0	2.0, 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)

2.1.2 Differences between TiDB Operator v2 and v1

With the rapid development of TiDB and the Kubernetes ecosystem, TiDB Operator releases v2, which is incompatible with v1. For a detailed comparison between v2 and v1, see Comparison Between TiDB Operator v2 and v1.

2.1.3 Manage TiDB clusters using TiDB Operator

In Kubernetes environments, you can use TiDB Operator to efficiently deploy and manage TiDB clusters. You can choose from the following deployment methods based on your requirements:

- To quickly deploy TiDB Operator and set up a TiDB cluster in a test environment, see Get Started with TiDB on Kubernetes.
- To deploy TiDB Operator with custom configurations, see Deploy TiDB Operator.

Before deploying in any environment, you can customize TiDB configurations using the following guides:

- Configure storage volumes
- Customize pods

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

- Deploy a TiDB Cluster
- Access a TiDB Cluster
- Scale a TiDB Cluster
- View TiDB Logs on Kubernetes

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.

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, see Deploy TiDB on Kubernetes.

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
4. Connect to TiDB

3.1 Step 1: Create a test Kubernetes cluster

This section describes how to create a local test Kubernetes cluster using kind. You can also refer to the Kubernetes official documentation for other deployment options.

kind lets you run a local Kubernetes cluster using containers as nodes. To install kind, see Quick Start.

The following uses kind v0.24.0 as an example:

```
kind create cluster --name tidb-operator
```

Expected output

```
create cluster with image kindest/node:v1.31.0@sha256:53
    ↪ df588e04085fd41ae12de0c3fe4c72f7013bba32a20e7325357a1ac94ba865
Creating cluster "tidb-operator" ...
   Ensuring node image (kindest/node:v1.31.0)
   Preparing nodes
   Writing configuration
   Starting control-plane
   Installing CNI
   Installing StorageClass
   Joining worker nodes
Set kubectl context to "kind-tidb-operator"
You can now use your cluster with:

kubectl cluster-info --context kind-tidb-operator

Have a question, bug, or feature request? Let us know! https://kind.sigs.k8s
    ↪ .io/#community
```

Check whether the cluster is successfully created:

```
kubectl cluster-info --context kind-tidb-operator
```

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'.
```

Now that your Kubernetes cluster is ready, you can deploy TiDB Operator.

3.2 Step 2: Deploy TiDB Operator

To deploy TiDB Operator, perform the following steps:

1. Install TiDB Operator CRDs
2. Install TiDB Operator

3.2.1 Install TiDB Operator CRDs

TiDB Operator includes multiple Custom Resource Definitions (CRDs) that implement different components of the TiDB cluster. Run the following command to install the CRDs in the cluster:

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.crds.yaml --server-side
```

3.2.2 Install TiDB Operator

Run the following command to install TiDB Operator into the cluster:

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.yaml --server-side
```

Check whether the TiDB Operator components are running normally:

```
kubectl get pods --namespace tidb-admin
```

Expected output

```
NAME                            READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m22s
```

When all pods are in the Running state, continue to the next step.

3.3 Step 3: Deploy a TiDB cluster

To deploy a TiDB cluster, perform the following steps:

1. Create a namespace:

> **Note:**
> Cross-namespace `Cluster` references are not supported. Make sure that all components are deployed in the same Kubernetes namespace.

```
kubectl create namespace db
```

2. Deploy the TiDB cluster.

   Method 1: use the following command to create a TiDB cluster that includes PD, TiKV, and TiDB components

   Create the `Cluster` resource:

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: basic
  namespace: db
```

```
kubectl apply -f cluster.yaml --server-side
```

   Create the PD component:

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 1
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
```

```
    - name: data
      mounts:
      - type: data
      storage: 20Gi
```

```
kubectl apply -f pd.yaml --server-side
```

Create the TiKV component:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 1
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

```
kubectl apply -f tikv.yaml --server-side
```

Create the TiDB component:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: tidb
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 1
  template:
    metadata:
      annotations:
```

```
    author: pingcap
  spec:
    version: v8.5.2
```

```
kubectl apply -f tidb.yaml --server-side
```

Method 2: save the preceding YAML files to a local directory and deploy the TiDB cluster with a single command:

```
kubectl apply -f ./<directory> --server-side
```

3. Monitor the status of Pod:

```
watch kubectl get pods -n db
```

Expected output

```
NAME              READY   STATUS    RESTARTS   AGE
pd-pd-68t96d      1/1     Running   0          2m
tidb-tidb-coqwpi  1/1     Running   0          2m
tikv-tikv-sdoxy4  1/1     Running   0          2m
```

After all component Pods start, each component type (`pd`, `tikv`, and `tidb`) will be in the Running state. You can press Ctrl+C to return to the command line and proceed to the next step.

3.4 Step 4: Connect to TiDB

Because TiDB supports the MySQL transport protocol and most of its syntax, you can use the `mysql` command-line tool to connect to TiDB directly. The following steps describe how to connect to the TiDB cluster.

3.4.1 Install the `mysql` command-line tool

To connect to TiDB, you need to install a MySQL-compatible command-line client on the machine where you are running `kubectl`. You can install MySQL Server, MariaDB Server, or Percona Server MySQL executables, or install them from your operating system's software repository.

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, list services in the `db` namespace:

```
kubectl get svc -n db
```

Expected output

```
NAME           TYPE       CLUSTER-IP     EXTERNAL-IP PORT(S)              AGE
pd-pd          ClusterIP  10.96.229.12   <none>      2379/TCP,2380/TCP    3m
pd-pd-peer     ClusterIP  None           <none>      2379/TCP,2380/TCP    3m
tidb-tidb      ClusterIP  10.96.174.237  <none>      4000/TCP,10080/TCP   3m
tidb-tidb-peer ClusterIP  None           <none>      10080/TCP            3m
tikv-tikv-peer ClusterIP  None           <none>      20160/TCP,20180/TCP  3m
```

In this example, the TiDB service is `tidb-tidb`.

Then, use the following command to forward a local port to the cluster:

```
kubectl port-forward -n db svc/tidb-tidb 14000:4000 > pf14000.out &
```

If port `14000` is already in use, you can use a different available port. The command runs in the background and forwards output to the file `pf14000.out`, so you can continue to run commands 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 MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 178505
Server version: 8.0.11-TiDB-v8.5.2 TiDB Server (Apache License 2.0)
    ↪ Community Edition, MySQL 8.0 compatible

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

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

```
MySQL [(none)]>
```

Use the following sample commands to verify the cluster is working.

Create a hello_world table

```
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: 18
              START_KEY: 7480000000000000FF6800000000000000F8
                END_KEY: 748000FFFFFFFFFFFF900000000000000F8
               TABLE_ID: 104
                DB_NAME: test
             TABLE_NAME: hello_world
               IS_INDEX: 0
               INDEX_ID: NULL
             INDEX_NAME: NULL
           IS_PARTITION: 0
           PARTITION_ID: NULL
         PARTITION_NAME: NULL
         EPOCH_CONF_VER: 5
          EPOCH_VERSION: 57
          WRITTEN_BYTES: 0
             READ_BYTES: 0
       APPROXIMATE_SIZE: 1
       APPROXIMATE_KEYS: 0
  REPLICATIONSTATUS_STATE: NULL
REPLICATIONSTATUS_STATEID: NULL
1 row in set (0.015 sec)
```

Query the TiDB version

```
mysql> select tidb_version()\G
*************************** 1. row ***************************
tidb_version(): Release Version: v8.5.2
Edition: Community
Git Commit Hash: 945d07c5d5c7a1ae212f6013adfb187f2de24b23
Git Branch: HEAD
UTC Build Time: 2024-05-21 03:51:57
```

```
GoVersion: go1.21.10
Race Enabled: false
Check Table Before Drop: false
Store: tikv
1 row in set (0.001 sec)
```

4 Deploy

4.1 Deploy TiDB Operator on Kubernetes

This document describes how to deploy TiDB Operator on Kubernetes.

4.1.1 Prerequisites

Before deploying TiDB Operator, make sure your environment meets the following software requirements:

- Kubernetes >= v1.30
- kubectl >= v1.30
- Helm >= v3.8

4.1.2 Deploy a Kubernetes cluster

TiDB Operator runs in a Kubernetes cluster. You can set up a Kubernetes cluster using one of the following options:

- **Self-managed cluster**: set up a self-managed Kubernetes cluster using any method described in the Kubernetes official documentation.
- **Cloud provider**: use a Kubernetes service provided by a Kubernetes certified cloud provider.

Whichever option you choose, make sure your Kubernetes version is v1.30 or later. To quickly create a simple cluster for testing, see Get Started.

4.1.3 Deploy TiDB Operator CRDs

Run the following command to install the Custom Resource Definitions (CRDs) required by TiDB Operator:

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.crds.yaml --server-side
```

4.1.4 Deploy TiDB Operator

You can deploy TiDB Operator using either of the following methods:

- Method 1: Deploy using `kubectl apply`
- Method 2: Deploy using Helm

4.1.4.1 Method 1: Deploy using `kubectl apply`

All resources required to install TiDB Operator (except CRDs), including RBAC and Deployment objects, are packaged in the `tidb-operator.yaml` file. You can deploy everything with a single command:

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ v2.0.0-beta.0/tidb-operator.yaml --server-side
```

TiDB Operator will be deployed in the `tidb-admin` namespace. To verify the deployment, run:

```
kubectl get pods -n tidb-admin
```

Expected output:

```
NAME                            READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m
```

4.1.4.2 Method 2: Deploy using Helm

Use Helm to deploy all resources except CRDs:

```
helm install tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator --
    ↪ version v2.0.0-beta.0 --namespace tidb-admin --create-namespace
```

TiDB Operator will be deployed in the `tidb-admin` namespace. To verify the deployment, run:

```
kubectl get pods -n tidb-admin
```

Expected output:

```
NAME                            READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m
```

4.1.4.2.1 Customize the deployment

To customize deployment parameters, first export the default `values.yaml` file:

```
helm show values oci://ghcr.io/pingcap/charts/tidb-operator --version v2
    ↪ .0.0-beta.0 > values.yaml
```

Edit the `values.yaml` file as needed, then install TiDB Operator with the customized settings:

```
helm install tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator --
    ↪ version v2.0.0-beta.0 -f values.yaml
```

4.2 Deploy a TiDB Cluster on Kubernetes

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

4.2.1 Prerequisites

- TiDB Operator is deployed.

4.2.2 Configure the TiDB cluster

A TiDB cluster consists of the following components. Each component is managed by a corresponding Custom Resource Definition (CRD):

Component	CRD
PD	`PDGroup`
TiKV	`TiKVGroup`
TiDB	`TiDBGroup`
TiProxy (optional)	`TiProxyGroup`
TiFlash (optional)	`TiFlashGroup`
TiCDC (optional)	`TiCDCGroup`

In the following steps, you will define a TiDB cluster using the `Cluster` CRD. Then, in each component CRD, specify the `cluster.name` field to associate the component with the cluster.

```
spec:
  cluster:
    name: <cluster>
```

Before deploying the cluster, prepare a YAML file for each component. The following lists some example configurations:

- PD: pd.yaml
- TiKV: tikv.yaml
- TiDB: tidb.yaml
- TiFlash: tiflash.yaml
- TiProxy: tiproxy.yaml
- TiCDC: ticdc.yaml

4.2.2.1 Configure component version

Use the `version` field to specify the component version:

```
spec:
  template:
    spec:
      version: v8.5.2
```

To use a custom image, set the `image` field:

```
spec:
  template:
    spec:
      version: v8.5.2
      image: gcr.io/xxx/tidb
```

If the version does not follow semantic versioning, you can specify it using the `image` field:

```
spec:
  template:
    spec:
      version: v8.5.2
      image: gcr.io/xxx/tidb:dev
```

> **Note:**
>
> TiDB Operator determines upgrade dependencies between components based on the `version` field. To avoid upgrade failures, ensure the image version is correct.

4.2.2.2 Configure resources

Use the `spec.resources` field to define the CPU and memory resources for a component:

```
spec:
  resources:
    cpu: "4"
    memory: 8Gi
```

Note:

- By default, the same values apply to both requests and limits.
- To set different values for requests and limits, use the Overlay feature.

4.2.2.3 Configure component parameters

Use the `spec.config` field to define `config.toml` settings:

```
spec:
  config: |
    [log]
    level = warn
```

Note:

Validation of `config.toml` content is not currently supported. Make sure your configuration is correct.

4.2.2.4 Configure volumes

Use the `spec.volumes` field to define mounted volumes for a component:

```
spec:
  template:
    spec:
      volumes:
      - name: test
        mounts:
        - mountPath: "/test"
        storage: 100Gi
```

Some components support a `type` field to specify a volume's purpose. Related fields in `config.toml` are updated automatically. For example:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
...
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        # data is for TiKV's data dir
        - type: data
        storage: 100Gi
```

You can also specify a StorageClass and VolumeAttributeClass. For details, see Volume Configuration.

4.2.2.5 Configure scheduling policies

Use the `spec.schedulePolicies` field to distribute components evenly across nodes:

```
spec:
  schedulePolicies:
  - type: EvenlySpread
    evenlySpread:
      topologies:
      - topology:
          topology.kubernetes.io/zone: us-west-2a
      - topology:
          topology.kubernetes.io/zone: us-west-2b
      - topology:
          topology.kubernetes.io/zone: us-west-2c
```

To assign weights to topologies, use the `weight` field:

```
spec:
  schedulePolicies:
  - type: EvenlySpread
    evenlySpread:
      topologies:
      - weight: 2
        topology:
          topology.kubernetes.io/zone: us-west-2a
      - topology:
          topology.kubernetes.io/zone: us-west-2b
```

You can also configure the following scheduling options using the Overlay feature:

- NodeSelector
- Toleration
- Affinity
- TopologySpreadConstraints

4.2.3 Deploy the TiDB cluster

After preparing the YAML files for each component, deploy the TiDB cluster by following these steps:

1. Create a namespace:

> **Note:**
> Cross-namespace references for `Cluster` resources are not supported. Make sure to deploy all components in the same namespace.

```
kubectl create namespace db
```

2. Deploy the TiDB cluster:

   Option 1: Deploy each component individually. The following example shows how to deploy a TiDB cluster with PD, TiKV, and TiDB.

   The following is an example configuration for the `Cluster` CRD:

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: basic
  namespace: db
```

   Create the `Cluster` CRD:

```
kubectl apply -f cluster.yaml --server-side
```

   The following is an example configuration for the PD component:

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
  namespace: db
spec:
  cluster:
    name: basic
```

```
  replicas: 3
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 20Gi
```

Create the PD component:

```
kubectl apply -f pd.yaml --server-side
```

The following is an example configuration for the TiKV component:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 3
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

Create the TiKV component:

```
kubectl apply -f tikv.yaml --server-side
```

The following is an example configuration for the TiDB component:

```
apiVersion: core.pingcap.com/v1alpha1
```

```
kind: TiDBGroup
metadata:
  name: tidb
  namespace: db
spec:
  cluster:
    name: basic
  replicas: 2
  template:
    metadata:
      annotations:
        author: pingcap
    spec:
      version: v8.5.2
```

Create the TiDB component:

```
kubectl apply -f tidb.yaml --server-side
```

Option 2: Deploy all components at once. You can save all component YAML files in a local directory and execute the following command:

```
kubectl apply -f ./<directory> --server-side
```

3. Check the status of the TiDB cluster:

```
kubectl get cluster -n db
kubectl get group -n db
```

4.3 Access the TiDB Cluster on Kubernetes

This document describes how to access a TiDB cluster through a Kubernetes Service. You can configure the Service as one of the following types, depending on your access requirements:

- `ClusterIP`: for access from within the Kubernetes cluster only.
- `NodePort`: for access from outside the cluster (recommended for test environments).
- `LoadBalancer`: for access through your cloud provider's LoadBalancer feature (recommended for production environments).

4.3.1 ClusterIP

The `ClusterIP` Service type exposes the TiDB cluster using an internal IP address. It is only accessible from within the Kubernetes cluster.

You can access the TiDB cluster using one of the following DNS formats:

- `basic-tidb`: access is limited to the same namespace.
- `basic-tidb.default`: support cross-namespace access.
- `basic-tidb.default.svc`: support cross-namespace access.

In these formats, `basic-tidb` is the Service name, and `default` is the namespace. For more information, see DNS for Services and Pods.

Each TiDBGroup automatically creates a Service that provides access to all TiDB instances in that group. For example, the TiDBGroup `tidb-0` creates an internal Service named `tidb-0-tidb`.

> **Note:**
>
> It is not recommended to directly use the default Service to access TiDB. Instead, create custom Services based on your specific needs.

The following YAML example defines a Service that provides access to all TiDB nodes in the `db` cluster:

```
apiVersion: v1
kind: Service
metadata:
  name: tidb
spec:
  selector:
    pingcap.com/managed-by: tidb-operator
    pingcap.com/cluster: db
    pingcap.com/component: tidb
  ports:
    - name: mysql
      protocol: TCP
      port: 4000
      targetPort: mysql-client
```

The following YAML example defines a Service that provides access to all TiDB nodes in the TiDBGroup `tidb-0` of the cluster `db`:

```
apiVersion: v1
kind: Service
metadata:
  name: tidb-0
spec:
  selector:
```

```
    pingcap.com/managed-by: tidb-operator
    pingcap.com/cluster: db
    pingcap.com/component: tidb
    pingcap.com/group: tidb-0
  ports:
    - name: mysql
      protocol: TCP
      port: 4000
      targetPort: mysql-client
```

4.3.2 NodePort

In environments without a LoadBalancer, you can use a `NodePort` Service to expose TiDB outside the Kubernetes cluster. This allows access using the node's IP address and a specific port. For more information, see NodePort.

> **Note:**
>
> It is not recommended to use `NodePort` in production environments. For production environments on cloud platforms, use the `LoadBalancer` type instead.

The following is an example:

```
apiVersion: v1
kind: Service
metadata:
  name: tidb-0
spec:
  type: NodePort
  selector:
    pingcap.com/managed-by: tidb-operator
    pingcap.com/cluster: db
    pingcap.com/component: tidb
    pingcap.com/group: tidb-0
  ports:
    - name: mysql
      protocol: TCP
      port: 4000
      targetPort: mysql-client
```

4.3.3 LoadBalancer

On cloud platforms that support LoadBalancer (such as Google Cloud or AWS), it is recommended to use the platform's LoadBalancer feature to expose TiDB. This approach provides higher availability and better load balancing.

For more information, see the following documents:

- AWS Load Balancer Controller
- Google Cloud LoadBalancer Service
- Azure Load Balancer Service

To learn more about Kubernetes Service types and cloud provider support for LoadBalancer, see the Kubernetes Service documentation.

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 how to configure alerts for the cluster.

5.1.1 Monitor the TiDB cluster

TiDB cluster monitoring consists of two parts: monitoring data and dashboards. You can collect metrics using open-source tools such as Prometheus or VictoriaMetrics, and display the metrics using Grafana.

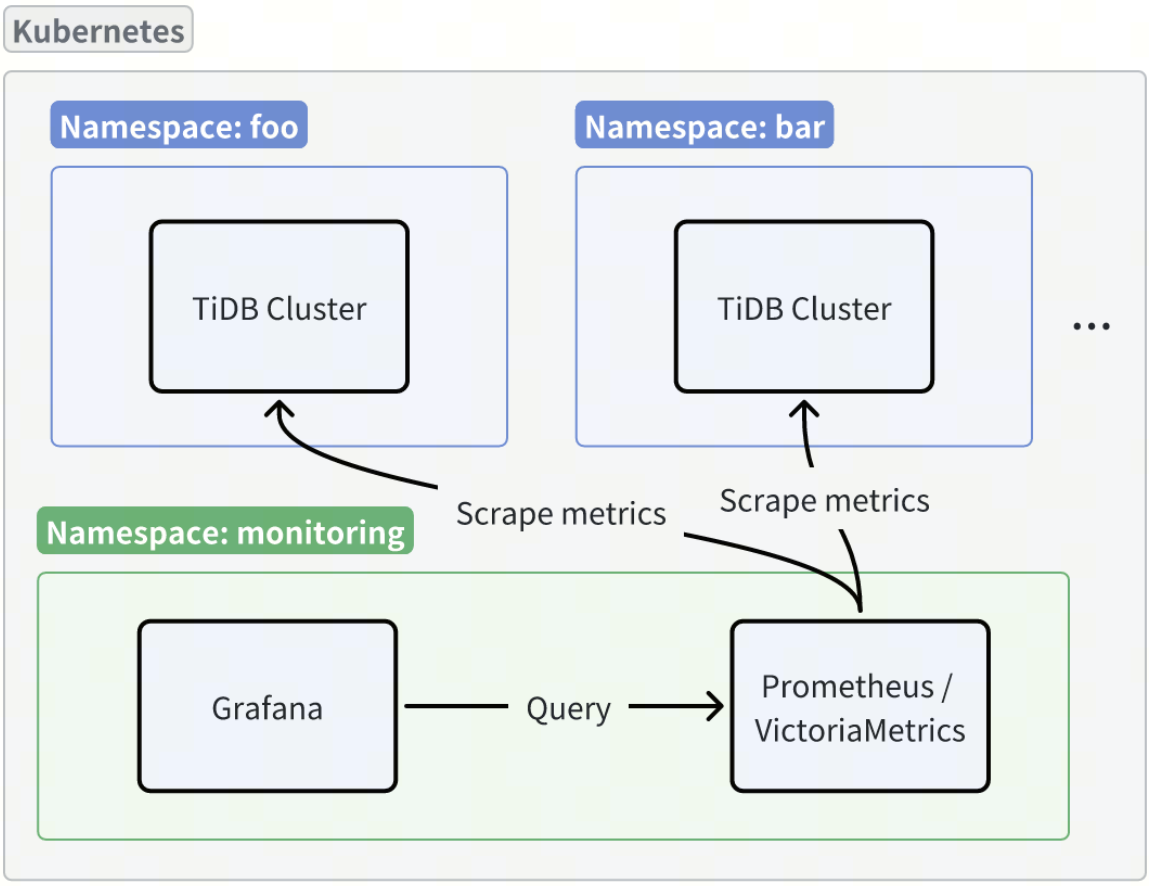

Figure 1: Monitoring architecture of TiDB clusters

5.1.1.1 Collect monitoring data

5.1.1.1.1 Collect monitoring data using Prometheus

To collect monitoring data using Prometheus, perform the following steps:

1. Deploy Prometheus Operator in your Kubernetes cluster by following the Prometheus Operator official documentation. This document uses version `v0.82.0` as an example.
2. Create a `PodMonitor` Custom Resource (CR) in the namespace of your TiDB cluster:

```
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
  name: tidb-cluster-pod-monitor
  namespace: ${tidb_cluster_namespace}
```

```
  labels:
    monitor: tidb-cluster
spec:
  jobLabel: "pingcap.com/component"
  namespaceSelector:
    matchNames:
      - ${tidb_cluster_namespace}
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: tidb-operator
  podMetricsEndpoints:
    - interval: 15s
      # If TLS is enabled in the TiDB cluster, set the scheme to https.
        ↪ Otherwise, set it to http.
      scheme: https
      honorLabels: true
      # Configure tlsConfig only if TLS is enabled in the TiDB cluster.
      tlsConfig:
        ca:
          secret:
            name: db-cluster-client-secret
            key: ca.crt
        cert:
          secret:
            name: db-cluster-client-secret
            key: tls.crt
        keySecret:
          name: db-cluster-client-secret
          key: tls.key
      metricRelabelings:
        - action: labeldrop
          regex: container
      relabelings:
        - sourceLabels: [
            ↪ __meta_kubernetes_pod_annotation_prometheus_io_scrape]
          action: keep
          regex: "true"
        - sourceLabels:
            - __meta_kubernetes_pod_name
            - __meta_kubernetes_pod_label_app_kubernetes_io_instance
            - __meta_kubernetes_pod_label_app_kubernetes_io_component
            - __meta_kubernetes_namespace
            - __meta_kubernetes_pod_annotation_prometheus_io_port
          action: replace
          regex: (.+);(.+);(.+);(.+);(.+)
```

```
      replacement: $1.$2-$3-peer.$4:$5
      targetLabel: __address__
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_annotation_prometheus_io_path]
      targetLabel: __metrics_path__
    - sourceLabels: [__meta_kubernetes_namespace]
      targetLabel: kubernetes_namespace
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_instance]
      targetLabel: cluster
    - sourceLabels: [__meta_kubernetes_pod_name]
      targetLabel: instance
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_component]
      targetLabel: component
    - sourceLabels:
        - __meta_kubernetes_namespace
        - __meta_kubernetes_pod_label_app_kubernetes_io_instance
      separator: '-'
      targetLabel: tidb_cluster
```

3. Create a `Prometheus` CR to collect metrics. Follow the Prometheus Operator official documentation and make sure the appropriate permissions are granted to the ServiceAccount:

```
apiVersion: monitoring.coreos.com/v1
kind: Prometheus
metadata:
  name: prometheus
  namespace: monitoring
spec:
  serviceAccountName: prometheus
  externalLabels:
    k8s_cluster: ${your_k8s_cluster_name}
  podMonitorSelector:
    matchLabels:
      monitor: tidb-cluster
  # An empty podMonitorNamespaceSelector means PodMonitors in all
    ↪ namespaces are collected.
  podMonitorNamespaceSelector: {}
```

4. Execute the following `kubectl port-forward` command to access Prometheus through port forwarding:

```
kubectl port-forward -n monitoring prometheus-prometheus-0 9090:9090
  ↪ &>/tmp/portforward-prometheus.log &
```

Then, you can access http://localhost:9090/targets in your browser view the monitoring data collection status.

5.1.1.1.2 Collect monitoring data using VictoriaMetrics

To collect monitoring data using VictoriaMetrics, perform the following steps:

1. Deploy VictoriaMetrics Operator in your Kubernetes cluster by following the VictoriaMetrics official documentation. This document uses version `v0.58.1` as an example.
2. Create a `VMSingle` Custom Resource (CR) to store monitoring data:

```
apiVersion: victoriametrics.com/v1beta1
kind: VMSingle
metadata:
  name: demo
  namespace: monitoring
```

3. Create a `VMAgent` CR to collect monitoring data:

```
apiVersion: victoriametrics.com/v1beta1
kind: VMAgent
metadata:
  name: demo
  namespace: monitoring
spec:
  # Configure remoteWrite to write collected monitoring metrics to
    ↪ VMSingle.
  remoteWrite:
    - url: "http://vmsingle-demo.monitoring.svc:8429/api/v1/write"
  externalLabels:
    k8s_cluster: ${your_k8s_cluster_name}
  selectAllByDefault: true
```

4. Create a `VMPodScrape` CR in the TiDB cluster namespace to discover Pods and generate scrape configs for VMAgent:

```
apiVersion: victoriametrics.com/v1beta1
kind: VMPodScrape
metadata:
  name: tidb-cluster-pod-scrape
  namespace: ${tidb_cluster_namespace}
spec:
  jobLabel: "pingcap.com/component"
```

```
namespaceSelector:
  matchNames:
    - ${tidb_cluster_namespace}
selector:
  matchLabels:
    app.kubernetes.io/managed-by: tidb-operator
podMetricsEndpoints:
  - interval: 15s
    # If TLS is enabled in the TiDB cluster, set the scheme to https.
       ↪ Otherwise, set it to http.
    scheme: https
    honorLabels: true
    # Configure tlsConfig only if TLS is enabled in the TiDB cluster.
    tlsConfig:
      ca:
        secret:
          name: db-cluster-client-secret
          key: ca.crt
      cert:
        secret:
          name: db-cluster-client-secret
          key: tls.crt
      keySecret:
        name: db-cluster-client-secret
        key: tls.key
    metricRelabelConfigs:
      - action: labeldrop
        regex: container
    relabelConfigs:
      - sourceLabels: [
          ↪ __meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: "true"
      - sourceLabels:
          - __meta_kubernetes_pod_name
          - __meta_kubernetes_pod_label_app_kubernetes_io_instance
          - __meta_kubernetes_pod_label_app_kubernetes_io_component
          - __meta_kubernetes_namespace
          - __meta_kubernetes_pod_annotation_prometheus_io_port
        action: replace
        regex: (.+);(.+);(.+);(.+);(.+)
        replacement: $1.$2-$3-peer.$4:$5
        targetLabel: __address__
      - sourceLabels: [
          ↪ __meta_kubernetes_pod_annotation_prometheus_io_path]
```

```
      targetLabel: __metrics_path__
    - sourceLabels: [__meta_kubernetes_namespace]
      targetLabel: kubernetes_namespace
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_instance]
      targetLabel: cluster
    - sourceLabels: [__meta_kubernetes_pod_name]
      targetLabel: instance
    - sourceLabels: [
        ↪ __meta_kubernetes_pod_label_app_kubernetes_io_component]
      targetLabel: component
    - sourceLabels:
        - __meta_kubernetes_namespace
        - __meta_kubernetes_pod_label_app_kubernetes_io_instance
      separator: '-'
      targetLabel: tidb_cluster
```

5. Execute the following `kubectl port-forward` command to access VMAgent through port forwarding:

```
kubectl port-forward -n monitoring svc/vmagent-demo 8429:8429 &>/tmp/
    ↪ portforward-vmagent.log &
```

Then, you can access http://localhost:8429/targets in your browser view the monitoring data collection status.

5.1.1.2 Configure monitoring dashboards

To configure the monitoring dashboard, perform the following steps:

1. Follow the Grafana official documentation to deploy Grafana. This document uses version `12.0.0-security-01` as an example.

2. Execute the following `kubectl port-forward` command to access Grafana through port forwarding:

```
kubectl port-forward -n ${namespace} ${grafana_pod_name} 3000:3000 &>/
    ↪ tmp/portforward-grafana.log &
```

3. Then, you can access http://localhost:3000 in your browser. The default username and password are both `admin`. If you install Grafana using Helm, execute the following command to get the password of `admin`:

```
kubectl get secret --namespace ${namespace} ${grafana_secret_name} -o
    ↪ jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

4. Add a data source of type Prometheus in Grafana and set the Prometheus Server URL based on your monitoring setup:

    - For Prometheus, set the URL to `http://prometheus-operated.monitoring.svc:9090`.
    - For VictoriaMetrics, set the URL to `http://vmsingle-demo.monitoring.svc:8429`.

5. Download Grafana dashboards for TiDB components using the `get-grafana-dashboards.sh` script and import them manually into Grafana.

5.1.2 Configure alerts

You can manage and send alerts using Alertmanager. For specific deployment and configuration steps, refer to the Alertmanager official documentation.

5.1.3 Monitor multiple clusters using Grafana

To monitor multiple clusters in Grafana, perform the following steps:

1. In the Grafana dashboard, click **Dashboard settings** to open the **Settings** page.
2. On the **Settings** page, select the **tidb_cluster** variable under **Variables**, and set the **Hide** property of the **tidb_cluster** variable to empty.
3. Return to the dashboard. You will see a cluster selector dropdown. Each option follows the `${namespace}-${tidb_cluster_name}` format.
4. Click **Save dashboard** to apply the changes.

5.2 Kubernetes Observability: Monitoring, Alerts, and Log Collection

This document describes how to monitor, configure alerts, and collect logs in a Kubernetes cluster. These practices help you assess the health and status of your cluster and its components.

5.2.1 Monitoring

5.2.1.1 Monitor TiDB components

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.

5.2.1.2 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.

The following lists some common open source monitoring systems that can be used to monitor server resources:

- Prometheus and node_exporter
- VictoriaMetrics
- CollectD
- Nagios
- Zabbix

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

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

5.2.1.3 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 might 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 using Prometheus Operator based on Node Exporter and Prometheus. This solution can also be compatible with and used for monitoring host resources.

5.2.2 Alerts

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.

5.2.3 Log collection

5.2.3.1 Collect TiDB and Kubernetes component runtime logs

When you deploy TiDB using TiDB Operator, all components write runtime logs the container's `stdout` and `stderr` by default. On Kubernetes, these logs are stored in the `/var/log/containers` directory on host machines, and filenames include the Pod and container names. You can collect application logs in the container directly from the host.

If you already have a log collection system in your existing infrastructure, you only need to add the `/var/log/containers/*.log` files from the Kubernetes hosts to your collection scope. If there is no log collection system available, or if you want to deploy a separate log collection system, then you can use any log collection system that you are familiar with.

The following lists some common open source tools for Kubernetes log collection:

- Vector
- Fluentd
- Fluent Bit
- Filebeat
- Logstash

You can typically aggregate collected logs and store them on a central server or in a dedicated storage and analysis system such as Elasticsearch.

Some cloud service providers or performance monitoring service providers also offer free or paid log collection solutions.

5.2.3.2 Collect system logs

You can collect system logs from Kubernetes hosts using standard methods. If you already have a log collection system in your existing infrastructure, you only need to add the relevant servers and log files to the collection scope. If there is no log collection system available, or if you want to deploy a separate log collection system, then you can use any log collection system that you are familiar with.

All tools listed in the Collect TiDB and Kubernetes component runtime logs section support system log collection. Additionally, some cloud service providers or performance monitoring service providers also offer free or paid log collection solutions.

6 Configure

6.1 Component Configuration

This document describes how to configure parameters for TiDB, TiKV, PD, TiProxy, TiFlash, and TiCDC in a Kubernetes cluster.

By default, TiDB Operator applies configuration changes by performing a rolling restart of the related components.

6.1.1 Configure TiDB parameters

You can configure TiDB parameters using the `spec.template.spec.config` field in the `TiDBGroup` CR.

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: tidb
spec:
  template:
    spec:
      config: |
        split-table = true
        oom-action = "log"
```

For a full list of configurable TiDB parameters, see TiDB Configuration File.

6.1.2 Configure TiKV parameters

You can configure TiKV parameters using the `spec.template.spec.config` field in the `TiKVGroup` CR.

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      config: |
        [storage]
          [storage.block-cache]
            capacity = "16GB"
        [log.file]
```

```
    max-days = 30
    max-backups = 30
```

For a full list of configurable TiKV parameters, see TiKV Configuration File.

Note:

The RocksDB logs of TiKV are stored in the `/var/lib/tikv` data directory by default. It is recommended to configure `max-days` and `max-backups` to automatically clean up log files.

6.1.3 Configure PD parameters

You can configure PD parameters using the `spec.template.spec.config` field in the `PDGroup` CR.

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      config: |
        lease = 3
        enable-prevote = true
```

For a full list of configurable PD parameters, see PD Configuration File.

Note:

After the cluster is started for the first time, some PD configuration items are persisted in etcd. The persisted configuration in etcd takes precedence over that in PD. Therefore, after the first start, you cannot modify some PD configuration using parameters. You need to dynamically modify the configuration using SQL statements, pd-ctl, or PD server API. Currently, among all the configuration items listed in Modify PD configuration online, except `log.level`, all the other configuration items cannot be modified using parameters after the first start.

6.1.3.1 Configure PD microservices

> **Note:**
>
> - Starting from v8.0.0, PD supports the microservice mode.
> - PD microservice mode can only be enabled during initial deployment and cannot be changed afterward.

To enable PD microservice mode, set `spec.template.spec.mode` to `"ms"` in the `PDGroup` CR:

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      mode: "ms"
```

Currently, PD supports the `tso` and `scheduling` microservices. You can configure them using the `TSOGroup` and `SchedulerGroup` CRs.

```
apiVersion: core.pingcap.com/v1alpha1
kind: TSOGroup
metadata:
  name: tso
spec:
  template:
    spec:
      config: |
        [log.file]
          filename = "/pdms/log/tso.log"
---
apiVersion: core.pingcap.com/v1alpha1
kind: SchedulerGroup
metadata:
  name: scheduling
spec:
  template:
    spec:
      config: |
        [log.file]
```

```
    filename = "/pdms/log/scheduling.log"
```

To get complete configuration parameters for the PD microservice, `tso` microservice, and `scheduling` microservice, see the following documents:

- PD Configuration File
- TSO Configuration File
- Scheduling Configuration File

Note:

- 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.
- 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.

6.1.4 Configure TiProxy parameters

You can configure TiProxy parameters using the `spec.template.spec.config` field in the `TiProxyGroup` CR.

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiProxyGroup
metadata:
  name: tiproxy
spec:
  template:
    spec:
      config: |
        [log]
          level = "info"
```

For a full list of configurable TiProxy parameters, see TiProxy Configuration File.

6.1.5 Configure TiFlash parameters

You can configure TiFlash parameters using the `spec.template.spec.config` field in the `TiFlashGroup` CR.

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiFlashGroup
metadata:
  name: tiflash
spec:
  template:
    spec:
      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 a full list of configurable TiFlash parameters, see TiFlash Configuration File.

6.1.6 Configure TiCDC startup parameters

You can configure TiCDC startup parameters using the `spec.template.spec.config` field in the `TiCDCGroup` CR.

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiCDCGroup
metadata:
  name: ticdc
spec:
  template:
    spec:
      config: |
        gc-ttl = 86400
        log-level = "info"
```

For a full list of configurable TiCDC startup parameters, see TiCDC Configuration File.

6.2 Storage Volume Configuration

This document describes how to configure storage volumes for TiDB cluster components in TiDB Operator and how to modify existing storage volumes.

6.2.1 Overview

In TiDB Operator, storage volumes provide persistent storage for TiDB cluster components. Components such as PD, TiKV, and TiFlash require configured volumes to store data. The structure for configuring a volume is as follows:

```
volumes:
- name: <volume-name>
  mounts:
  - type: <mount-type>
    mountPath: <mount-path>
    subPath: <sub-path>
  storage: <storage-size>
  storageClassName: <storage-class-name>
  volumeAttributesClassName: <volume-attributes-class-name>
```

Field descriptions:

- `name`: the name of the volume, which must be unique within the same component.
- `mounts`: defines the mount information for the volume, including:
  - `type`: specifies the mount type. Supported types vary by component.
  - `mountPath` (optional): specifies the mount path. If not specified, the default path is used.
  - `subPath` (optional): specifies the sub-path within the volume.
- `storage`: specifies the storage capacity, such as `100Gi`.
- `storageClassName` (optional): specifies the name of the Kubernetes storage class.
- `volumeAttributesClassName` (optional): specifies the volume attributes class for modifying volume attributes. This feature is supported in Kubernetes 1.29 and later versions.

6.2.2 Component-specific volume configuration

6.2.2.1 PD storage volume configuration

Supported mount types for PD:

- `data`: PD data directory. The default path is `/var/lib/pd`.

Example:

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
```

```
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 20Gi
```

6.2.2.2 TiKV storage volume configuration

Supported mount types for TiKV:

- `data`: TiKV data directory. The default path is `/var/lib/tikv`.

Example:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

6.2.2.3 TiDB storage volume configuration

Supported mount types for TiDB:

- `data`: TiDB data directory. The default path is `/var/lib/tidb`.
- `slowlog`: TiDB slow log directory. The default path is `/var/log/tidb`.

Example:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TidbGroup
metadata:
  name: tidb
spec:
  template:
```

```
  spec:
    volumes:
    - name: slowlog
      mounts:
        - type: slowlog
      storage: 10Gi
```

6.2.2.4 TiFlash storage volume configuration

Supported mount types for TiFlash:

- `data`: TiFlash data directory. The default path is `/var/lib/tiflash`.

Example:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiFlashGroup
metadata:
  name: tiflash
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
```

6.2.3 Modify storage volumes

6.2.3.1 Modify the storage size

By modifying the `volumes.storage` field in the component group CR, TiDB Operator automatically updates the corresponding PVC to adjust the storage size.

Note:

- You can only modify storage size when the `allowVolumeExpansion` setting of the StorageClass in use is set to `true`.
- Only volume scale-out is supported. Scale-in is not supported.

6.2.3.2 Change volume attributes

For TiDB clusters that use cloud provider storages, TiDB Operator supports the following two methods to modify storage volume attributes (such as IOPS and throughput):

- (Recommended) Kubernetes native method
- Cloud provider API

6.2.3.2.1 Method 1: Kubernetes native

The Kubernetes native method uses the Volume Attributes Classes feature to modify volume attributes.

Prerequisites:

- Kubernetes 1.29 or later versions.
- The Volume Attributes Classes feature of Kubernetes is enabled.

To modify volume attributes using the Kubernetes native method, perform the following:

1. Enable the `VolumeAttributesClass` feature in the `TidbCluster` CR:

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: basic
spec:
  featureGates:
    - name: VolumeAttributesClass
```

2. Create a `VolumeAttributesClass` resource:

```
apiVersion: storage.k8s.io/v1beta1
kind: VolumeAttributesClass
metadata:
  name: silver
driverName: pd.csi.storage.gke.io
parameters:
  provisioned-iops: "3000"
  provisioned-throughput: "50"
```

3. Modify volume attributes by modifying the `volumes.volumeAttributesClassName` field in the component group CR to use the `VolumeAttributesClass` created in step 2:

```
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
        volumeAttributesClassName: silver
```

6.2.3.2.2 Method 2: cloud provider API

> **Note:**
>
> When using the cloud provider API method, you need to configure the corresponding cloud provider permissions for TiDB Operator.

When the `VolumeAttributesClass` feature is not enabled, TiDB Operator calls cloud provider APIs directly to modify storage volume attributes. Currently, modifications to the following cloud provider storage volumes are supported:

- **AWS EBS**: modify EBS volume size, IOPS, and throughput using the AWS EC2 API.
- **Azure Disk**: modify Managed Disk size, IOPS, and throughput using the Azure API.

Using AWS EBS as an example, assume that the current storage volume configuration is:

```
spec:
  template:
    spec:
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
        storageClassName: gp3-2000-100
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
```

```
  name: gp3-2000-100
parameters:
  csi.storage.k8s.io/fstype: ext4
  encrypted: "true"
  iops: "2000"
  throughput: "100"
  type: gp3
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

You can create a StorageClass with higher IOPS and throughput:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gp3-4000-400
parameters:
  csi.storage.k8s.io/fstype: ext4
  encrypted: "true"
  iops: "4000"
  throughput: "400"
  type: gp3
provisioner: ebs.csi.aws.com
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

Then, modify the `.spec.template.spec.volumes.storageClassName` field in the component group CR to `gp3-4000-400`. TiDB Operator automatically calls cloud provider APIs to perform the modification.

Because the StorageClass of a PVC cannot be modified directly, TiDB Operator adds the following annotations to the PVC to track the update:

- `spec.tidb.pingcap.com/revision`: desired spec revision number.
- `spec.tidb.pingcap.com/storage-class`: desired storage class.
- `spec.tidb.pingcap.com/storage-size`: desired storage size.
- `status.tidb.pingcap.com/revision`: current spec revision number.
- `status.tidb.pingcap.com/storage-class`: current storage class.
- `status.tidb.pingcap.com/storage-size`: current storage size.

6.2.4 FAQ

6.2.4.1 How do I configure different storage sizes for different TiKV instances?

All instances in the same `TiKVGroup` share the same storage configuration. To configure different storage for different TiKV instances, you can create multiple `TiKVGroups` resources.

6.2.4.2 Why does the volume modification not take effect immediately?

Storage volume modifications might not take effect immediately for the following reasons:

- Some cloud providers have cooldown period restrictions for volume modifications. For example, AWS EBS has a 6-hour cooldown period.
- File system expansion might take some time to complete.

6.3 Customize the Configuration of Kubernetes Native Resources

This document describes how to use the Overlay feature of TiDB Operator to customize Kubernetes native resource configurations.

The Overlay feature is a configuration mechanism in TiDB Operator that enables you to customize the configuration of native resources in Kubernetes clusters (such as Pods and PersistentVolumeClaims (PVCs)) without modifying the TiDB Operator source code. This enables you to meet specific deployment requirements.

6.3.1 Supported resource types

Currently, TiDB Operator supports customizing the following resource types through the Overlay feature:

- **Pod**: modify the Pod metadata (such as labels and annotations) and the `spec` field.
- **PersistentVolumeClaim (PVC)**: modify PVC metadata (such as labels and annotations).

> **Note:**
>
> It is not supported to modify the `spec` field of a PVC.

6.3.2 Usage

6.3.2.1 Customize Pod configurations (Pod Overlay)

Pod Overlay enables you to modify the Pod metadata (such as labels and annotations) and the `spec` field. You can configure this using the `spec.template.spec.overlay.pod` field in the Custom Resource (CR) of a component group (such as `PDGroup`, `TiDBGroup`, `TiKVGroup`, `TiFlashGroup`, `TiProxyGroup`, or `TiCDCGroup`).

The following example shows how to add an environment variable named `CUSTOM_ENV_VAR` ↪ to the PD container:

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: pd
                env:
                  - name: "CUSTOM_ENV_VAR"
                    value: "custom_value"
```

6.3.2.2 Customize PVC configurations (PVC Overlay)

PVC Overlay enables you to modify the PVC metadata (such as labels and annotations) and the `spec` field. You can configure this using the `spec.template.spec.overlay.` ↪ `volumeClaims` field in the Custom Resource (CR) of a component group (such as `PDGroup` ↪ , `TiDBGroup`, `TiKVGroup`, `TiFlashGroup`, `TiProxyGroup`, or `TiCDCGroup`).

The following example shows how to add a custom label named `custom-label` to the PVC of the TiKV component:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      volumes:
      - name: data
```

```
      mounts:
        - type: data
      storage: 100Gi
    overlay:
      volumeClaims:
        - name: data
          volumeClaim:
            metadata:
              labels:
                custom-label: "value"
```

6.3.3 Notes

When using the Overlay feature, note the following:

- The Overlay feature merges the configuration you define in the `spec.template.spec.overlay` field with the default configuration automatically generated by TiDB Operator. If conflicts exist, the configuration defined in Overlay takes precedence.
- For map-type fields such as `nodeSelector` and `labels`, the Overlay feature appends the key-value pairs you define while preserving existing configurations, rather than replacing them entirely.
- Before modifying configurations using the Overlay feature, ensure that you fully understand the potential impact of these changes, especially for critical configurations such as `securityContext` and resource limits.
- The fields that can be modified through Overlay depend on the Kubernetes API version used by TiDB Operator.

6.3.4 Example use cases

This section provides common examples of using the Overlay feature to help you understand and apply it for customizing Kubernetes resource configurations.

6.3.4.1 Configure Pod resource limits and affinity

The following example shows how to use Overlay to configure resource request limits and affinity for TiDB Pods:

```
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
```

```
      - name: tidb
        resources:
          limits:
            cpu: "4"
            memory: "8Gi"
          requests:
            cpu: "2"
            memory: "4Gi"
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: dedicated
              operator: In
              values:
              - tidb
```

6.3.4.2 Configure Pod security context

The following example shows how to configure Pod `sysctls` parameters using Overlay:

```
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            securityContext:
              sysctls:
              - name: net.core.somaxconn
                value: "1024"
```

6.3.4.3 Update Pod and PVC labels or annotations in place

Using Overlay, you can update Pod and PVC labels or annotations without restarting the Pod:

```
spec:
  template:
    spec:
      overlay:
        pod:
          metadata:
            labels:
```

```
            custom-label: "value"
          annotations:
            custom-annotation: "value"
      volumeClaims:
        - name: data
          volumeClaim:
            metadata:
              labels:
                custom-label: "value"
              annotations:
                custom-annotation: "value"
```

6.3.4.4 Inject a sidecar container

You can use Overlay to inject a sidecar container into a Pod for purposes such as monitoring or log collection:

```
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            initContainers:
              - name: logshipper
                image: alpine:latest
                restartPolicy: Always
                command: ['sh', '-c', 'tail -F /opt/logs.txt']
```

7 Manage

7.1 Security

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. To enable TLS for the MySQL client, perform the following steps:

1. Issue two sets of certificates: a set of server-side certificates for the TiDB server, and a set of client-side certificates for the MySQL client. Create two Secret objects, `${tidb_group_name}-tidb-server-secret` and `${tidb_group_name}-tidb-client-secret`, respectively including these two sets of certificates.

Note:

- The Secret objects you created must follow the preceding naming convention. Otherwise, the deployment of the TiDB cluster will fail.
- Explicitly specifying the MySQL TLS Secret will be supported in a future release.
- The default naming convention for Secrets differs between TiDB Operator v2 and v1:
  - For TiDB clusters created by TiDB Operator v1, the default Secret names are `${cluster_name}-tidb-server-secret` and `${cluster_name}-tidb-client-secret`.
  - In TiDB Operator v2, different `TiDBGroup` objects support different TLS certificates. Therefore, the default Secret names are `${tidb_group_name}-tidb-server-secret` and `${tidb_group_name}-tidb-client-secret`.

2. Deploy the cluster, and set the `.spec.template.spec.security.tls.mysql.enabled` field in `TiDBGroup` to `true`.

Note:

Enabling or modifying the TLS configuration of a running `TiDBGroup` triggers a rolling restart of TiDB Pods. Perform this operation with caution.

3. Configure the MySQL client to use an encrypted connection.

There are multiple ways to issue certificates. This document provides two methods, and you can also issue certificates for TiDB clusters as needed. The two methods are:

- Use the `cfssl` system to issue certificates
- (Recommended) Use the `cert-manager` system to issue certificates

To renew existing TLS certificates, see Renew and Replace the TLS Certificate.

7.1.1.1 Step 1: Issue two sets of certificates for the TiDB cluster

7.1.1.1.1 Use `cfssl` to issue certificates

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` and `hosts` attributes:

```
...
    "CN": "TiDB Server",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${tidb_group_name}-tidb",
      "${tidb_group_name}-tidb.${namespace}",
      "${tidb_group_name}-tidb.${namespace}.svc",
      "*.${tidb_group_name}-tidb",
      "*.${tidb_group_name}-tidb.${namespace}",
      "*.${tidb_group_name}-tidb.${namespace}.svc",
      "*.${tidb_group_name}-tidb-peer",
      "*.${tidb_group_name}-tidb-peer.${namespace}",
      "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
    ],
...
```

`${tidb_group_name}` is the name of `TiDBGroup`. `${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` and `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 ${tidb_group_name}-tidb-server-secret --
       ↪ namespace=${namespace} --from-file=tls.crt=server.pem --from-file
       ↪ =tls.key=server-key.pem --from-file=ca.crt=ca.pem
   kubectl create secret generic ${tidb_group_name}-tidb-client-secret --
       ↪ namespace=${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 the other 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.

7.1.1.1.2 Use `cert-manager` to issue certificates

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 to save 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}-cert-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 the Issuer of the 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: ${tidb_group_name}-tidb-server-secret
  namespace: ${namespace}
spec:
  secretName: ${tidb_group_name}-tidb-server-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
  dnsNames:
    - "${tidb_group_name}-tidb"
    - "${tidb_group_name}-tidb.${namespace}"
    - "${tidb_group_name}-tidb.${namespace}.svc"
    - "*.${tidb_group_name}-tidb"
    - "*.${tidb_group_name}-tidb.${namespace}"
    - "*.${tidb_group_name}-tidb.${namespace}.svc"
    - "*.${tidb_group_name}-tidb-peer"
    - "*.${tidb_group_name}-tidb-peer.${namespace}"
    - "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
  ipAddresses:
    - 127.0.0.1
    - ::1
  issuerRef:
```

```
    name: ${cluster_name}-cert-issuer
    kind: Issuer
    group: cert-manager.io
```

`${cluster_name}` is the name of the cluster. `${tidb_group_name}` is the name of `TiDBGroup`:

- Set `spec.secretName` to `${tidb_group_name}-tidb-server-secret`
- Add `server auth` in `usages`.
- Add the following six DNSs in `dnsNames`. You can also add other DNSs according to your needs:
- `${tidb_group_name}-tidb`
- `${tidb_group_name}-tidb.${namespace}`
- `${tidb_group_name}-tidb.${namespace}.svc`
- `*.${tidb_group_name}-tidb`
- `*.${tidb_group_name}-tidb.${namespace}`
- `*.${tidb_group_name}-tidb.${namespace}.svc`
- `*.${tidb_group_name}-tidb-peer`
- `*.${tidb_group_name}-tidb-peer.${namespace}`
- `*.${tidb_group_name}-tidb-peer.${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 preceding created Issuer 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 `${tidb_group_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: ${tidb_group_name}-tidb-client-secret
  namespace: ${namespace}
spec:
  secretName: ${tidb_group_name}-tidb-client-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
```

```
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - client auth
  issuerRef:
    name: ${cluster_name}-cert-issuer
    kind: Issuer
    group: cert-manager.io
```

`${cluster_name}` is the name of the cluster. `${tidb_group_name}` is the name of `TiDBGroup`:

- Set `spec.secretName` to `${tidb_group_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 `${tidb_group_name}-tidb-client-secret` Secret object to be used by the TiDB client.

Note:

- The `ca.crt` included in the Secret issued by cert-manager is the CA that signed the certificate, not the CA used to validate the peer's mTLS certificate.
- In this example, the client and server TLS certificates are issued by the same CA, so they can be used directly. If the client and server certificates are issued by different CAs, it is recommended to use the Trust Manager to distribute the appropriate `ca.crt`.

7.1.1.2 Step 2: Deploy the TiDBGroup

The following configuration example shows how to create a `TiDBGroup` with MySQL TLS enabled:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
```

```
  name: tidb
spec:
  cluster:
    name: tls
  version: v8.5.2
  replicas: 1
  template:
    spec:
      security:
        tls:
          mysql:
            enabled: true
      config: |
        [security]
        cluster-verify-cn = ["TiDB"]
```

7.1.1.3 Step 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 TLS connections.

Execute the following command to acquire the client-side certificate and connect to the TiDB server:

```
kubectl get secret -n ${namespace} ${tidb_group_name}-tidb-client-secret -
    ↪ ojsonpath='{.data.tls\.crt}' | base64 --decode > client-tls.crt
kubectl get secret -n ${namespace} ${tidb_group_name}-tidb-client-secret -
    ↪ ojsonpath='{.data.tls\.key}' | base64 --decode > client-tls.key
kubectl get secret -n ${namespace} ${tidb_group_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
```

Finally, to verify whether TLS is successfully enabled, refer to Check whether the current connection uses encryption.

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. The steps are as follows:

1. Generate certificates for each component group of the TiDB cluster to be created:

Create a separate set of certificates for each component group, and save it as a Kubernetes Secret object named `${group_name}-${component_name}-cluster-secret`.

> **Note:**
> The Secret objects you created must follow the preceding naming convention. Otherwise, the deployment of the TiDB components will fail.

2. Deploy the cluster and set the `.spec.tlsCluster.enabled` field to `true` in the Cluster Custom Resource (CR).

> **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.

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:

- Use `cfssl`
- Use `cert-manager`

If you need to renew the existing TLS certificate, refer to Renew and Replace the TLS Certificate.

7.1.2.1 Step 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 Use `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:

Note:

- All TiDB components share the same set of TLS certificates for inter-component communication to encrypt traffic between clients and servers. Therefore, when generating the CA configuration, you must specify both `server auth` and `client auth`.
- It is recommended that all component certificates be issued by the same CA.

```
cat << EOF > ca-config.json
{
    "signing": {
        "default": {
            "expiry": "8760h"
        },
        "profiles": {
            "internal": {
                "expiry": "8760h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "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 certificates:

   In this step, you need to generate a set of certificates for each component group of the TiDB cluster.

   - PD certificate

     First, generate the default `pd.json` file:

     ```
     cfssl print-defaults csr > pd.json
     ```

     Then, edit this file to change the `CN` and `hosts` attributes:

     ```
     ...
        "CN": "TiDB",
        "hosts": [
          "127.0.0.1",
          "::1",
          "${pd_group_name}-pd",
          "${pd_group_name}-pd.${namespace}",
          "${pd_group_name}-pd.${namespace}.svc",
          "${pd_group_name}-pd-peer",
          "${pd_group_name}-pd-peer.${namespace}",
          "${pd_group_name}-pd-peer.${namespace}.svc",
          "*.${pd_group_name}-pd-peer",
          "*.${pd_group_name}-pd-peer.${namespace}",
          "*.${pd_group_name}-pd-peer.${namespace}.svc"
        ],
     ```

```
...
```

`${pd_group_name}` is the name of PDGroup, and `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

Finally, generate the PD certificate:

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
    ↪ -profile=internal pd.json | cfssljson -bare pd
```

- TiKV certificate

  First, generate the default `tikv.json` file:

  ```
  cfssl print-defaults csr > tikv.json
  ```

  Then, edit this file to change the `CN` and `hosts` attributes:

  ```
  ...
      "CN": "TiDB",
      "hosts": [
        "127.0.0.1",
        "::1",
        "${tikv_group_name}-tikv",
        "${tikv_group_name}-tikv.${namespace}",
        "${tikv_group_name}-tikv.${namespace}.svc",
        "${tikv_group_name}-tikv-peer",
        "${tikv_group_name}-tikv-peer.${namespace}",
        "${tikv_group_name}-tikv-peer.${namespace}.svc",
        "*.${tikv_group_name}-tikv-peer",
        "*.${tikv_group_name}-tikv-peer.${namespace}",
        "*.${tikv_group_name}-tikv-peer.${namespace}.svc"
      ],
  ...
  ```

  `${tikv_group_name}` is the name of TiKVGroup, and `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

  Finally, generate the TiKV certificate:

  ```
  cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
      ↪ -profile=internal tikv.json | cfssljson -bare tikv
  ```

- TiDB certificate

  First, generate the default `tidb.json` file:

  ```
  cfssl print-defaults csr > tidb.json
  ```

  Then, edit this file to change the `CN` and `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${tidb_group_name}-tidb",
      "${tidb_group_name}-tidb.${namespace}",
      "${tidb_group_name}-tidb.${namespace}.svc",
      "${tidb_group_name}-tidb-peer",
      "${tidb_group_name}-tidb-peer.${namespace}",
      "${tidb_group_name}-tidb-peer.${namespace}.svc",
      "*.${tidb_group_name}-tidb-peer",
      "*.${tidb_group_name}-tidb-peer.${namespace}",
      "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
    ],
...
```

`${tidb_group_name}` is the name of TiDBGroup, and `${namespace}` is the namespace in which the TiDB cluster is deployed. You can also add your customized `hosts`.

Finally, generate the TiDB certificate:

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
    ↪ -profile=internal tidb.json | cfssljson -bare tidb
```

- Other components

  In addition to PD, TiKV, and TiDB, other component groups also require their own TLS certificates. The following example shows the basic steps to generate a component certificate:

  First, generate the default `${component_name}.json` file:

```
cfssl print-defaults csr > ${component_name}.json
```

  Then, edit this file to change the `CN` and `hosts` attributes:

```
...
    "CN": "TiDB",
    "hosts": [
      "127.0.0.1",
      "::1",
      "${group_name}-${component_name}",
      "${group_name}-${component_name}.${namespace}",
      "${group_name}-${component_name}.${namespace}.svc",
      "${group_name}-${component_name}-peer",
      "${group_name}-${component_name}-peer.${namespace}",
      "${group_name}-${component_name}-peer.${namespace}.svc",
```

```
      "*.${group_name}-${component_name}-peer",
      "*.${group_name}-${component_name}-peer.${namespace}",
      "*.${group_name}-${component_name}-peer.${namespace}.svc"
    ],
...
```

In this file:

- `${group_name}` is the name of the component group.
- `${component_name}` is the name of the component (use lowercase letters, such as `pd`, `tikv`, and `tidb`).
- `${namespace}` the namespace in which the TiDB cluster is deployed.
- You can also add your customized `hosts`.

Finally, generate the component certificate:

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json
    ↪ -profile=internal ${component_name}.json | cfssljson -bare $
    ↪ {component_name}
```

6. Create the Kubernetes Secret object:

   If you have already generated a set of certificates for each component and a set of client-side certificates for each client as described in the preceding steps, create the Secret objects for the TiDB cluster by executing the following command:

   Create the Secret for the PD cluster certificate:

```
kubectl create secret generic ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=pd.pem --from-file=tls
    ↪ .key=pd-key.pem --from-file=ca.crt=ca.pem
```

   Create the Secret for the TiKV cluster certificate:

```
kubectl create secret generic ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tikv.pem --from-file=
    ↪ tls.key=tikv-key.pem --from-file=ca.crt=ca.pem
```

   Create the Secret for the TiDB cluster certificate:

```
kubectl create secret generic ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${namespace} --from-file=tls.crt=tidb.pem --from-file=
    ↪ tls.key=tidb-key.pem --from-file=ca.crt=ca.pem
```

   Create the Secret for other component certificates:

```
kubectl create secret generic ${group_name}-${component_name}-cluster-
    ↪ secret --namespace=${namespace} --from-file=tls.crt=${
    ↪ component_name}.pem --from-file=tls.key=${component_name}-key.pem
    ↪  --from-file=ca.crt=ca.pem
```

In this step, separate Secrets are created for the server-side certificates of PD, TiKV, and TiDB for loading during startup, and another set of client-side certificates is provided for their client connections.

7.1.2.1.2 Use `cert-manager`

1. Install `cert-manager`.

   For more information, see cert-manager installation on Kubernetes.

2. Create an Issuer to issue certificates to the TiDB cluster.

   To configure `cert-manager`, create the Issuer resources.

   First, create a directory to save 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}-certs-issuer
```

```
  namespace: ${namespace}
spec:
  ca:
    secretName: ${cluster_name}-ca-secret
```

`${cluster_name}` is the name of the cluster. The preceding 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 component 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 Between TiDB Components, each component needs a certificate.

   - PD certificate

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${pd_group_name}-pd-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${pd_group_name}-pd-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${pd_group_name}-pd"
  - "${pd_group_name}-pd.${namespace}"
  - "${pd_group_name}-pd.${namespace}.svc"
```

```
  - "${pd_group_name}-pd-peer"
  - "${pd_group_name}-pd-peer.${namespace}"
  - "${pd_group_name}-pd-peer.${namespace}.svc"
  - "*.${pd_group_name}-pd-peer"
  - "*.${pd_group_name}-pd-peer.${namespace}"
  - "*.${pd_group_name}-pd-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-certs-issuer
    kind: Issuer
    group: cert-manager.io
```

`${pd_group_name}` is the name of PDGroup, and `${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${pd_group_name}-pd-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
- `${pd_group_name}-pd`
- `${pd_group_name}-pd.${namespace}`
- `${pd_group_name}-pd.${namespace}.svc`
- `${pd_group_name}-pd-peer`
- `${pd_group_name}-pd-peer.${namespace}`
- `${pd_group_name}-pd-peer.${namespace}.svc`
- `*.${pd_group_name}-pd-peer`
- `*.${pd_group_name}-pd-peer.${namespace}`
- `*.${pd_group_name}-pd-peer.${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 preceding created Issuer in `issuerRef`.
- For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${pd_group_name}-pd-cluster-secret` Secret object to be used by the PD component of the TiDB cluster.

- TiKV certificate

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tikv_group_name}-tikv-cluster-secret
  namespace: ${namespace}
```

```
spec:
  secretName: ${tikv_group_name}-tikv-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${tikv_group_name}-tikv"
  - "${tikv_group_name}-tikv.${namespace}"
  - "${tikv_group_name}-tikv.${namespace}.svc"
  - "${tikv_group_name}-tikv-peer"
  - "${tikv_group_name}-tikv-peer.${namespace}"
  - "${tikv_group_name}-tikv-peer.${namespace}.svc"
  - "*.${tikv_group_name}-tikv-peer"
  - "*.${tikv_group_name}-tikv-peer.${namespace}"
  - "*.${tikv_group_name}-tikv-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-certs-issuer
    kind: Issuer
    group: cert-manager.io
```

`${tikv_group_name}` is the name of TiKVGroup, and `${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${tikv_group_name}-tikv-cluster-secret`.
- Add `server auth` and `client auth` in `usages`.
- Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
- `${tikv_group_name}-tikv`
- `${tikv_group_name}-tikv.${namespace}`
- `${tikv_group_name}-tikv.${namespace}.svc`
- `${tikv_group_name}-tikv-peer`
- `${tikv_group_name}-tikv-peer.${namespace}`
- `${tikv_group_name}-tikv-peer.${namespace}.svc`
- `*.${tikv_group_name}-tikv-peer`
- `*.${tikv_group_name}-tikv-peer.${namespace}`
- `*.${tikv_group_name}-tikv-peer.${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 preceding created Issuer in `issuerRef`.
  - For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${tikv_group_name}-tikv-cluster-secret` Secret object to be used by the TiKV component of the TiDB server.

- TiDB certificate

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${tidb_group_name}-tidb-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${tidb_group_name}-tidb-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${tidb_group_name}-tidb"
  - "${tidb_group_name}-tidb.${namespace}"
  - "${tidb_group_name}-tidb.${namespace}.svc"
  - "${tidb_group_name}-tidb-peer"
  - "${tidb_group_name}-tidb-peer.${namespace}"
  - "${tidb_group_name}-tidb-peer.${namespace}.svc"
  - "*.${tidb_group_name}-tidb-peer"
  - "*.${tidb_group_name}-tidb-peer.${namespace}"
  - "*.${tidb_group_name}-tidb-peer.${namespace}.svc"
  ipAddresses:
  - 127.0.0.1
  - ::1
  issuerRef:
    name: ${cluster_name}-certs-issuer
    kind: Issuer
    group: cert-manager.io
```

`${tidb_group_name}` is the name of TiDBGroup, and `${cluster_name}` is the

name of the cluster. Configure the items as follows:

  - Set `spec.secretName` to `${tidb_group_name}-tidb-cluster-secret`.
  - Add `server auth` and `client auth` in `usages`.
  - Add the following DNSs in `dnsNames`. You can also add other DNSs according to your needs:
  - `${tidb_group_name}-tidb`
  - `${tidb_group_name}-tidb.${namespace}`
  - `${tidb_group_name}-tidb.${namespace}.svc`
  - `${tidb_group_name}-tidb-peer`
  - `${tidb_group_name}-tidb-peer.${namespace}`
  - `${tidb_group_name}-tidb-peer.${namespace}.svc`
  - `*.${tidb_group_name}-tidb-peer`
  - `*.${tidb_group_name}-tidb-peer.${namespace}`
  - `*.${tidb_group_name}-tidb-peer.${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 preceding created Issuer in `issuerRef`.
  - For other attributes, refer to cert-manager API.

  After the object is created, `cert-manager` generates a `${tidb_group_name}-tidb-cluster-secret` Secret object to be used by the TiDB component of the TiDB server.

- Other component:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ${group_name}-${component_name}-cluster-secret
  namespace: ${namespace}
spec:
  secretName: ${group_name}-${component_name}-cluster-secret
  duration: 8760h # 365d
  renewBefore: 360h # 15d
  subject:
    organizations:
    - PingCAP
  commonName: "TiDB"
  usages:
    - server auth
    - client auth
  dnsNames:
  - "${group_name}-${component_name}"
  - "${group_name}-${component_name}.${namespace}"
```

```
- "${group_name}-${component_name}.${namespace}.svc"
- "${group_name}-${component_name}-peer"
- "${group_name}-${component_name}-peer.${namespace}"
- "${group_name}-${component_name}-peer.${namespace}.svc"
- "*.${group_name}-${component_name}-peer"
- "*.${group_name}-${component_name}-peer.${namespace}"
- "*.${group_name}-${component_name}-peer.${namespace}.svc"
ipAddresses:
- 127.0.0.1
- ::1
issuerRef:
  name: ${cluster_name}-certs-issuer
  kind: Issuer
  group: cert-manager.io
```

`${group_name}` is the name of the component group, `${component_name}` is the name of the component, and `${cluster_name}` is the name of the cluster. Configure the items as follows:

- Set `spec.secretName` to `${group_name}-${component_name}-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:
- `${group_name}-${component_name}`
- `${group_name}-${component_name}.${namespace}`
- `${group_name}-${component_name}.${namespace}.svc`
- `${group_name}-${component_name}-peer`
- `${group_name}-${component_name}-peer.${namespace}`
- `${group_name}-${component_name}-peer.${namespace}.svc`
- `*.${group_name}-${component_name}-peer`
- `*.${group_name}-${component_name}-peer.${namespace}`
- `*.${group_name}-${component_name}-peer.${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 preceding created Issuer in `issuerRef`.
- For other attributes, refer to cert-manager API.

After the object is created, `cert-manager` generates a `${group_name}-${component_name}-cluster-secret` Secret object to be used by the component of the TiDB server.

7.1.2.2 Step 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:

- For TiDB v8.3.0 and earlier versions, the PD configuration item `cert-allowed-cn` can only be set to a single value. Therefore, the `Common Name` of all authentication objects must be set to the same value.
- Starting from TiDB v8.4.0, the PD configuration item `cert-allowed-cn` supports multiple values. You can configure multiple `Common Name` in the `cluster-verify-cn` configuration item for TiDB and in the `cert-allowed-cn` configuration item for other components as needed.
- For more information, see Enable TLS Between TiDB Components.

Perform the following steps to create a TiDB cluster and enable TLS between TiDB components:

Create the `tidb-cluster.yaml` file:

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  tlsCluster:
    enabled: true
---
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: ${pd_group_name}
  namespace: ${namespace}
spec:
  cluster:
    name: ${cluster_name}
  version: v8.5.2
  replicas: 3
  template:
    spec:
      config: |
        [security]
```

```
      cert-allowed-cn = ["TiDB"]
    volumes:
    - name: data
      mounts:
      - type: data
      storage: 20Gi
---
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: ${tikv_group_name}
  namespace: ${namespace}
spec:
  cluster:
    name: ${cluster_name}
  version: v8.5.2
  replicas: 3
  template:
    spec:
      config: |
        [security]
        cert-allowed-cn = ["TiDB"]
      volumes:
      - name: data
        mounts:
        - type: data
        storage: 100Gi
---
apiVersion: core.pingcap.com/v1alpha1
kind: TiDBGroup
metadata:
  name: ${tidb_group_name}
  namespace: ${namespace}
spec:
  cluster:
    name: ${cluster_name}
  version: v8.5.2
  replicas: 1
  template:
    spec:
      config: |
        [security]
        cluster-verify-cn = ["TiDB"]
```

Then, execute `kubectl apply -f tidb-cluster.yaml` to create a TiDB cluster.

7.1.3 Run Containers as a Non-Root User

In some Kubernetes environments, containers cannot be run as the root user. For security reasons, it is recommended to run containers as a non-root user in production environments to reduce the risk of potential attacks. This document describes how to configure containers to run as a non-root user using the `securityContext`.

7.1.3.1 Configure containers related to TiDB Operator

For TiDB Operator containers, configure the `securityContext` in the Helm `values.yaml` file.

The following is an example configuration:

```
controllerManager:
  securityContext:
    runAsUser: 1000
    runAsGroup: 2000
    fsGroup: 2000
```

7.1.3.2 Configure containers generated by CR

For containers generated by Custom Resources (CRs), configure the `securityContext` in any CR, such as `PDGroup`, `TiDBGroup`, `TiKVGroup`, `TiFlashGroup`, `TiCDCGroup`, `Backup`, `CompactBackup`, `BackupSchedule`, or `Restore`.

- For CRs such as `PDGroup`, `TiDBGroup`, `TiKVGroup`, `TiFlashGroup`, and `TiCDCGroup`, configure the `securityContext` using the Overlay method. The following is an example configuration for the `PDGroup` CR:

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            securityContext:
              runAsUser: 1000
              runAsGroup: 2000
              fsGroup: 2000
```

- For CRs such as `Backup`, `CompactBackup`, `BackupSchedule`, and `Restore`, configure the `podSecurityContext` in the `spec` field. The following is an example configuration for the `Backup` CR:

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup
spec:
  podSecurityContext:
    runAsUser: 1000
    runAsGroup: 2000
    fsGroup: 2000
```

7.1.4 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.

7.1.4.1 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.

7.1.4.1.1 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 object based on the combined CA certificate.

```
kubectl create secret generic ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${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 ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${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 ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${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 -
```

In the preceding command, `${pd_group_name}`, `${tikv_group_name}`, and `${ ↪ tidb_group_name}` are the names of the component groups, and `${namespace}` is the namespace in which the TiDB cluster is deployed.

> **Note:**
>
> The preceding 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.

7.1.4.1.2 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
```

Note:

- The preceding 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 preceding 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 ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${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 ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${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 ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${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 -
```

In the preceding command, `${pd_group_name}`, `${tikv_group_name}`, and `${ ↪ tidb_group_name}` are the names of the component groups, and `${namespace}` is the namespace in which the TiDB cluster is deployed.

Note:

The preceding 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 uses 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.

7.1.4.1.3 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 ${pd_group_name}-pd-cluster-secret --
    ↪ namespace=${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 ${tikv_group_name}-tikv-cluster-secret --
    ↪ namespace=${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 ${tidb_group_name}-tidb-cluster-secret --
    ↪ namespace=${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 -
```

   In the preceding command, `${pd_group_name}`, `${tikv_group_name}`, and `${tidb_group_name}` are the names of the component groups, and `${namespace}` is the namespace in which the TiDB cluster is deployed.

> **Note:**
>
> - The preceding 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.

7.1.4.2 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.

7.1.4.2.1 Renew and replace the CA certificate

7.1.4.2.2 Only renew and replace certificates between components

When using cert-manager to issue certificates, you can configure the `spec.renewBefore` field of the `Certificate` resource to have cert-manager automatically renew the certificate before it expires.

1. cert-manager supports automatic renewal of component certificates and their corresponding Kubernetes Secret objects before expiration. To renew manually, refer to Renew certificates using cmctl.

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 automatically, 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.

7.2 Manually Scale TiDB on Kubernetes

This document introduces how to horizontally and vertically scale a TiDB cluster on Kubernetes.

7.2.1 Horizontal scaling

Horizontal scaling refers to increasing or decreasing the number of Pods in a component to scale the cluster. You can control the number of Pods by modifying the value of `replicas` of a certain component to scale out or scale in.

- To scale out a TiDB cluster, **increase** the value of `replicas` of a certain component. The scaling out operations add Pods 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 until the number of Pods equals the value of `replicas`.

To scale a TiDB cluster horizontally, use `kubectl` to modify the `spec.replicas` field in the corresponding Component Group Custom Resource (CR) object to the desired value.

1. Modify the `replicas` value of a component as needed. For example, configure the `replicas` value of PD to `3`:

```
kubectl patch -n ${namespace} pdgroup ${name} --type merge --patch '{"
    ↪ spec":{"replicas":3}}'
```

2. Verify that the Component Group CR for the corresponding component in the Kubernetes cluster has been updated to the expected configuration. For example, run the following command to check the PDGroup CR:

```
kubectl get pdgroup ${name} -n ${namespace}
```

   The `DESIRED` value in the output should match the value you have configured.

3. Check whether the number of Pods has increased or decreased:

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

   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.

   PD and TiDB components usually take 10 to 30 seconds to scale in or out.

   TiKV components usually take 3 to 5 minutes to scale in or out because the process involves data migration.

Note:

- 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 `Removing` state by running `kubectl get -n ${namespace} tikv`.
- When the number of `Serving` TiKV is equal to or less than the value of the `MaxReplicas` parameter in the PD configuration, the TiKV components cannot be scaled in.
- The TiKV component does not support scaling 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.

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.

To vertically scale up or scale down components including PD, TiKV, TiDB, TiProxy, TiFlash, and TiCDC, use `kubectl` to modify `spec.template.spec.resources` in the Component Group CR object that corresponds to the cluster to desired values.

> **Note:**
>
> Currently, In-Place Pod Resize is not supported.

7.2.2.1 View the vertical scaling progress

To view the upgrade progress of the cluster, run the following command:

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

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, TiFlash, and TiCDC 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 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 TiDB Operator

This document describes how to upgrade TiDB Operator to a specific version.

7.3.1.1 Before you begin

It is not supported to upgrade TiDB Operator from v1.x to v2.x.

7.3.1.2 Upgrade CRDs

To upgrade the Custom Resource Definitions (CRDs) for TiDB Operator, run the following command. Replace `${version}` with your target TiDB Operator version, such as `v2.0.0-beta.0`:

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ ${version}/tidb-operator.crds.yaml --server-side
```

7.3.1.3 Upgrade TiDB Operator components

You can upgrade TiDB Operator components using one of the following methods:

- Method 1: use `kubectl apply`
- Method 2: use Helm

7.3.1.3.1 Method 1: Upgrade using `kubectl apply`

To upgrade TiDB Operator components, run the following command:

```
kubectl apply -f https://github.com/pingcap/tidb-operator/releases/download/
    ↪ ${version}/tidb-operator.yaml --server-side
```

This command upgrades TiDB Operator deployed in the `tidb-admin` namespace. To verify that the upgrade is successful, run the following command:

```
kubectl get pods -n tidb-admin
```

Example output:

```
NAME                            READY   STATUS    RESTARTS   AGE
tidb-operator-6c98b57cc8-ldbnr 1/1   Running   0          2m
```

7.3.1.3.2 Method 2: Upgrade using Helm

If you deploy TiDB Operator using Helm, you can upgrade it using the `helm upgrade` command.

To upgrade TiDB Operator, run the following command:

```
helm upgrade tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator --
    ↪ version=${version} --namespace=tidb-admin
```

In the preceding command:

- `tidb-operator`: the Helm release name for TiDB Operator. Replace it if you use a different name.
- `${version}`: the target TiDB Operator version, such as `v2.0.0-beta.0`.
- `--namespace=tidb-admin`: the namespace where TiDB Operator is deployed. Replace it with your actual namespace if different.

After the upgrade is complete, you can check the Pod status with the following command to verify that the upgrade is successful:

```
kubectl get pods -n tidb-admin
```

Upgrade with a custom configuration

If you previously used a custom configuration during deployment or previous upgrades (that is, you modified the `values.yaml` file), make sure to use these custom configurations during this upgrade. Perform the following steps:

1. Export the `values.yaml` file used by the current deployment:

   ```
   helm get values tidb-operator -n tidb-admin > values.yaml
   ```

2. Get the default configuration file `values-new.yaml` for the target version:

   ```
   helm show values oci://ghcr.io/pingcap/charts/tidb-operator --version=$
       ↪ {version} > values-new.yaml
   ```

3. Compare the `values.yaml` and `values-new.yaml` files and merge your custom configuration items into `values-new.yaml`.

4. Use the updated `values-new.yaml` file to perform the upgrade:

   ```
   helm upgrade tidb-operator oci://ghcr.io/pingcap/charts/tidb-operator
       ↪ --version=${version} -f values-new.yaml --namespace=tidb-admin
   ```

7.3.2 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.

7.3.2.1 Rolling update introduction

Kubernetes provides the rolling update feature to update your application with zero downtime.

When you perform a rolling update, TiDB Operator waits for the new version of a Pod to run successfully before proceeding to 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 causes 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 `ADMIN SHOW DDL [JOBS|JOB QUERIES]` to confirm that there are no DDL operations in progress.

7.3.2.2 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.

7.3.2.3 Upgrade steps

1. Update the version of each component group in the cluster by setting the target `version` in the version field. For example:

```
spec:
  template:
    spec:
      version: v8.5.2
```

You can use the `kubectl apply` command to update all components at once, or use `kubectl edit` to update each component individually. TiDB Operator automatically handles the upgrade order and prevents the upgrade from continuing if the preconditions are not met.

> **Note:**
> TiDB Operator requires all components in the cluster to use the same version. Make sure that the `spec.template.spec.version` field for all components is set to the same version.

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.

7.4 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 TiB) 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

For more information, see the following documents:

- 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

7.4.1.1.2 Restore data

To recover the SST files exported by BR to a TiDB cluster, use BR. For more information, see the following documents:

- Restore Data from S3-Compatible Storage Using BR
- Restore Data from GCS Using BR
- Restore Data from Azure Blob Storage Using BR

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 set the value of `spec.cleanPolicy` to `Delete`, TiDB Operator cleans the backup data when it deletes the CR.

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.

In such cases, if you need to delete the namespace, it is recommended that you first delete all the `Backup` or `BackupSchedule` CRs and then delete the namespace.

If you delete the namespace before you delete the `Backup` or `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

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.

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`, `CompactBackup`, `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.toolImage`: the tool image used by `Backup`.

  - If the field is not specified or the value is empty, the `pingcap/br:${tikv_version}` image is used for backup by default.
  - If the BR version is specified in this field, such as `.spec.toolImage: pingcap/br:v8.5.2`, the image of the specified version is used for backup.
  - If an image is specified without the version, such as `.spec.toolImage: private/registry/br`, the `private/registry/br:${tikv_version}` image is used for backup.

- `.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`. This field has two value options currently:
  - `snapshot`: back up data through snapshots in the KV layer.
  - `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-stop`: permanently stops the log backup task. After executing this command, the Backup CR enters a stopped state and cannot be restarted.
  - `log-pause`: temporarily pauses the currently running log backup task. After pausing, you can use the `log-start` command to resume the task.
- `.spec.cleanPolicy`: The cleaning policy for the backup data when the backup CR is deleted. If this field is not configured, or if you configure a value other than the three policies above, the backup data is retained. You can choose one of 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.
- `.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.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. This field can be ignored by default. When the field is not configured, 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 to 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. 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 bucket name 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 CompactBackup CR fields

For TiDB v9.0.0 and later versions, you can use `CompactBackup` to accelerate PITR (Point-in-time recovery). To compact log backup data into structured SST files, you can create a custom `CompactBackup` CR object to define a backup task. The following introduces the fields in the `CompactBackup` CR:

- `.spec.startTs`: the start timestamp for log compaction backup.
- `.spec.endTs`: the end timestamp for log compaction backup.
- `.spec.concurrency`: the maximum number of concurrent log compaction tasks. The default value is `4`.
- `.spec.maxRetryTimes`: the maximum number of retries for failed compaction tasks. The default value is `6`.
- `.spec.toolImage`: the tool image used by `CompactBackup`. BR is the only tool image used in `CompactBackup`. When using BR for backup, you can specify the BR version with this field:

  - If not specified or left empty, the `pingcap/br:${tikv_version}` image is used for backup by default.
  - If a BR version is specified, such as `.spec.toolImage: pingcap/br:v9.0.0`, the image of the specified version is used for backup.
  - If an image is specified without a version, such as `.spec.toolImage: private ↪ /registry/br`, the `private/registry/br:${tikv_version}` image is used for backup.

- `.spec.env`: the environment variables for the Pod that runs the compaction task.
- `.spec.affinity`: the affinity configuration for the Pod that runs the compaction task. For details on affinity, refer to Affinity and anti-affinity.
- `.spec.tolerations`: specifies that the Pod that runs the compaction task 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 compaction task, 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 compaction task, 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 compaction task.
- `.spec.serviceAccount`: the name of the ServiceAccount used for compact.
- `.spec.useKMS`: whether to use AWS-KMS to decrypt the S3 storage key used for the backup.
- `.spec.br`: BR-related configuration. For more information, refer to BR fields.
- `.spec.s3`: S3-related configuration. For more information, refer to S3 storage fields.
- `.spec.gcs`: GCS-related configuration. For more information, refer to GCS fields.
- `.spec.azblob`: Azure Blob Storage-related configuration. For more information, refer to Azure Blob Storage fields.

7.4.2.3 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.toolImage`: the tools image used by `Restore`. For example, `spec.toolImage: ↪ pingcap/br:v8.5.2`. If not specified, `pingcap/br:${tikv_version}` 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.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, BR restores all the schemas in the backup file.

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.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.

7.4.2.4 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`.
- `compactBackupTemplate`: the configuration template of the log compaction backup. The fields are the same as those in the `spec` configuration of the `CompactBackup` CR. The compaction backup is created and deleted along with `backupSchedule`. The log backup names are stored in `status.logBackup`. The storage settings of the compaction backup should be the same as that of `logBackupTemplate` in the same `backupSchedule`.

> **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 Grant permissions to an AWS account

Amazon Web Services (AWS) provides different methods to grant permissions for different types of Kubernetes clusters. This document introduces the following three methods:

- Grant permissions by AccessKey and SecretKey: applicable to self-managed Kubernetes clusters and AWS EKS clusters.
- Grant permissions by associating IAM with Pod: applicable to self-managed Kubernetes clusters.
- Grant permissions by associating IAM with ServiceAccount: applicable only to AWS EKS clusters.

7.4.3.1.1 Grant permissions by AccessKey and SecretKey

To grant permissions to S3-compatible storage using AccessKey and SecretKey, perform the following steps:

1. Create an IAM user by following Create an IAM user in your AWS account and grant the required permissions. Because backup and restore operations require access to AWS S3 storage, grant the `AmazonS3FullAccess` permission to the IAM user.

2. Create an access key by following Create an access key for yourself (console). After completion, you can obtain the AccessKey and SecretKey.

3. Create a Kubernetes Secret named `s3-secret` using the following command and enter the AccessKey and SecretKey obtained in the previous step. This Secret stores the credentials required to access S3-compatible storage services.

```
kubectl create secret generic s3-secret --from-literal=access_key=<your
    ↪ -access-key> --from-literal=secret_key=<your-secret-key> --
    ↪ namespace=<your-namespace>
```

4. AWS clients support obtaining associated user permissions through the process environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. Therefore, you can grant pods access to S3-compatible storage services by setting the corresponding environment variables.

   The following example shows how to configure environment variables for TiKVGroup using Overlay (the configuration method for TiFlashGroup is the same):

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
  labels:
    pingcap.com/group: tikv
    pingcap.com/component: tikv
    pingcap.com/cluster: demo
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tikv
                env:
                  - name: "AWS_ACCESS_KEY_ID"
                    valueFrom:
                      secretKeyRef:
                        name: "s3-secret"
                        key: "access_key"
                  - name: "AWS_SECRET_ACCESS_KEY"
                    valueFrom:
                      secretKeyRef:
                        name: "s3-secret"
                        key: "secret_key"
```

   The following example shows how to configure environment variables for the Backup resource:

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup-s3
spec:
  env:
    - name: "AWS_ACCESS_KEY_ID"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
          key: "access_key"
    - name: "AWS_SECRET_ACCESS_KEY"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
          key: "secret_key"
```

The following example shows how to configure environment variables for the Restore resource:

```
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: restore-s3
spec:
  env:
    - name: "AWS_ACCESS_KEY_ID"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
          key: "access_key"
    - name: "AWS_SECRET_ACCESS_KEY"
      valueFrom:
        secretKeyRef:
          name: "s3-secret"
          key: "secret_key"
```

7.4.3.1.2 Grant permissions by associating IAM with Pod

The method of granting permissions by associating IAM with a Pod is supported by the open-source tool kube2iam. It enables processes within a Pod to inherit the permissions of an IAM role by associating the role with the Pod.

Note:

- kube2iam is only applicable to Kubernetes clusters running on AWS EC2 instances. It does not support other types of nodes.
- To use this authorization method, see the kube2iam documentation to set up the kube2iam environment in your Kubernetes cluster, and deploy TiDB Operator and the TiDB cluster.
- This method is not compatible with Pods that use the `hostNetwork` network mode.

To grant permissions by associating IAM with a Pod, perform the following steps:

1. Follow the IAM role creation document to create an IAM role in your AWS account and grant it the `AmazonS3FullAccess` policy.
2. Use the Overlay feature to associate the IAM role with the target component (TiKV or TiFlash). The following example shows how to associate the role with a TiKVGroup:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          annotations:
            iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
```

Note:

Replace `arn:aws:iam::123456789012:role/user` with the actual ARN of the IAM role you created in step 1.

7.4.3.1.3 Grant permissions by associating IAM with ServiceAccount

By associating a user's IAM role with a `ServiceAccount` resource in Kubernetes, any Pod using that ServiceAccount will inherit the permissions of the IAM role.

To grant permissions by associating IAM with a ServiceAccount, perform the following steps:

1. Follow the IAM role creation document to create an IAM role in your AWS account and grant it the `AmazonS3FullAccess` policy.

2. Follow the instructions in Create an IAM OIDC provider for your cluster to create an IAM OIDC provider for your EKS cluster.

3. Create a Kubernetes ServiceAccount named `br-s3`, and assign the IAM role to it as described in Assign IAM roles to Kubernetes service accounts.

4. Use the Overlay feature to associate the ServiceAccount with the Pod in the TiKV-Group or TiFlashGroup. The following example shows how to associate it in a TiKV-Group:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            serviceAccountName: br-s3
```

5. Specify the `serviceAccount` in the backup or restore configuration. The following example shows how to specify it in a Backup resource:

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup-s3
spec:
  serviceAccount: br-s3
```

7.4.3.2 Grant permissions to a Google Cloud account

7.4.3.2.1 Grant permissions by the service account

To grant permissions using a Google Cloud service account key, perform the following steps:

1. Follow the Create service accounts document to create a service account and generate a service account key file. Save the file as `google-credentials.json`.

2. Create a Kubernetes Secret named `gcp-secret` to store the credentials for accessing Google Cloud Storage:

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

3. Follow the instructions in Add a principal to a bucket-level policy to grant the service account created in step 1 access to the target storage bucket, and assign the `roles/storage.objectUser` role.

4. Set environment variables for the Pod. The following example shows how to configure it for a TiKVGroup:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tikv
                env:
                  - name: "GOOGLE_APPLICATION_CREDENTIALS"
                    valueFrom:
                      secretKeyRef:
                        name: "gcp-secret"
                        key: "credentials"
```

The following example shows how to configure the Backup resource to use the Secret:

```
```yaml
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: backup-gcp
spec:
  gcs:
    secretName: gcp-secret
```
```

The following example shows how to configure the Restore resource to use the Secret:

```yaml
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: restore-gcp
spec:
  gcs:
    secretName: gcp-secret
```

7.4.3.3 Grant permissions to an Azure account

Azure Blob Storage provides different methods to grant permissions for different types of Kubernetes clusters. This document introduces the following two methods:

- Grant permissions by access key: applicable to all types of Kubernetes clusters.
- Grant permissions by Azure AD: suitable for scenarios requiring fine-grained access control and key rotation.

7.4.3.3.1 Grant permissions by access key

Azure clients can read credentials from the environment variables `AZURE_STORAGE_ACCOUNT` and `AZURE_STORAGE_KEY`. To grant permissions using this method, perform the following steps:

1. Create a Kubernetes Secret named `azblob-secret` that stores the storage account name and key:

```
kubectl create secret generic azblob-secret \
  --from-literal=AZURE_STORAGE_ACCOUNT=<your-storage-account> \
  --from-literal=AZURE_STORAGE_KEY=<your-storage-key> \
  --namespace=<your-namespace>
```

2. Use the Overlay feature to inject the Secret as environment variables into the TiKVGroup or TiFlashGroup Pod. The following example shows how to configure it for a TiKVGroup:

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
```

```
overlay:
  pod:
    spec:
      containers:
        - name: tikv
          env:
            - name: "AZURE_STORAGE_ACCOUNT"
              valueFrom:
                secretKeyRef:
                  name: "azblob-secret"
                  key: "AZURE_STORAGE_ACCOUNT"
            - name: "AZURE_STORAGE_KEY"
              valueFrom:
                secretKeyRef:
                  name: "azblob-secret"
                  key: "AZURE_STORAGE_KEY"
```

7.4.3.3.2 Grant permissions by Azure AD

Azure clients can obtain access through the environment variables `AZURE_STORAGE_ACCOUNT` ↪ , `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_CLIENT_SECRET`. This method is ideal for higher security and automatic key rotation.

1. Create a Kubernetes Secret named `azblob-secret-ad` to store credentials for accessing Azure Blob Storage:

```
kubectl create secret generic azblob-secret-ad \
  --from-literal=AZURE_STORAGE_ACCOUNT=<your-storage-account> \
  --from-literal=AZURE_CLIENT_ID=<your-client-id> \
  --from-literal=AZURE_TENANT_ID=<your-tenant-id> \
  --from-literal=AZURE_CLIENT_SECRET=<your-client-secret> \
  --namespace=<your-namespace>
```

2. Use the Overlay feature to inject the Secret as environment variables into the TiKVGroup or TiFlashGroup Pod. The following example shows how to configure it for a TiKVGroup:

Note:

- When granting permissions by Azure AD, ensure the service principal has access to the target storage account.
- Restart the Pods after modifying the Secret to apply updated environment variables.

```
apiVersion: core.pingcap.com/v1alpha1
kind: TiKVGroup
metadata:
  name: tikv
spec:
  template:
    spec:
      overlay:
        pod:
          spec:
            containers:
              - name: tikv
                envFrom:
                  secretRef:
                    name: "azblob-secret-ad"
```

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 TiB) 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 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

Note:

- BR uses a fixed ServiceAccount name that must be `tidb-backup-manager`.
- Starting from TiDB Operator v2, the `apiGroup` for resources such as `Backup` and `Restore` changes from `pingcap.com` to `br.pingcap.com`.

1. Save the following content as the `backup-rbac.yaml` file to create the required role-based access control (RBAC) resources:

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. Execute the following command to create the RBAC resources in the `test1` namespace:

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. Grant permissions to the remote storage for the `test1` namespace:

- If you are using Amazon S3 to back up 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 back up your cluster, you can grant permissions by using AccessKey and SecretKey.

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 follows:

```
kubectl apply -f full-backup-s3.yaml
```

The content of `full-backup-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: test1
spec:
  backupType: full
  br:
    cluster: demo1
    # 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 follows:

```
kubectl apply -f full-backup-s3.yaml
```

The content of `full-backup-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  backupType: full
  br:
    cluster: demo1
    sendCredToTikv: false
    # 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 follows:

```
kubectl apply -f full-backup-s3.yaml
```

The content of `full-backup-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-s3
  namespace: test1
spec:
```

```
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
    # 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 test1 -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 no longer supported in TiDB Operator v2. It is recommended to use the `logSubcommand` field to ensure clear and consistent configuration.

Start log backup

1. In the `test1` 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: br.pingcap.com/v1alpha1
kind: Backup
```

```
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  br:
    cluster: demo1
    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 test1
```

```
NAME                                  COMPLETIONS  ...
backup-demo1-log-backup-s3-log-start 1/1          ...
```

3. View the newly created `Backup` CR:

```
kubectl get backup -n test1
```

```
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 test1
```

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 test1
```

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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    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 test1
```

```
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 test1
```

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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    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 test1
```

```
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 test1
```

Change the value of `logSubcommand` to `log-stop`, then save and exit the editor. The modified content is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    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 test1
```

```
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 test1
```

In the last line of the CR, append `spec.logTruncateUntil: "2022-10-10T15:21:00+08:00"`. Then save and exit the editor. The modified content is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-s3
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start/log-pause/log-stop
  br:
    cluster: demo1
    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 test1
```

```
NAME                                        COMPLETIONS  ...
...
backup-demo1-log-backup-s3-log-truncate 1/1      ...
```

3. View the `Backup` CR information:

```
kubectl describe backup -n test1
```

```
...
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 test1 -o wide
```

```
NAME                  MODE      STATUS    ...   LOGTRUNCATEUNTIL
demo1-log-backup-s3 log       Stopped   ...   2022-10-10T15:21:00+08:00
```

Compact log backup

For TiDB v9.0.0 and later versions, you can use a `CompactBackup` CR to compact log backup data into SST format, accelerating downstream PITR (Point-in-time recovery).

This section explains how to compact log backup based on the log backup example from previous sections.

1. In the `test1` namespace, create a `CompactBackup` CR named `demo1-compact-backup`.

```
kubectl apply -f compact-backup-demo1.yaml
```

The content of `compact-backup-demo1.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: CompactBackup
metadata:
  name: demo1-compact-backup
  namespace: test1
spec:
  startTs: "***"
  endTs: "***"
  concurrency: 8
  maxRetryTimes: 2
  br:
    cluster: demo1
    sendCredToTikv: true
  s3:
    provider: aws
    secretName: s3-secret
    region: us-west-1
    bucket: my-bucket
    prefix: my-log-backup-folder
```

The `startTs` and `endTs` fields specify the time range for the logs to be compacted by `demo1-compact-backup`. Any log that contains at least one write within this time range will be included in the compaction process. As a result, the final compacted data might include data written outside this range.

The `s3` settings should be the same as the storage settings of the log backup to be compacted. `CompactBackup` reads log files from the corresponding location and compacts them.

View the status of log backup compaction

After creating the `CompactBackup` CR, TiDB Operator automatically starts compacting the log backup. You can check the backup status using the following command:

```
kubectl get cpbk -n test1
```

From the output, you can find the status of the `CompactBackup` CR named `demo1-compact-backup`. An example output is as follows:

```
NAME                  STATUS                  PROGRESS
    ↪                                    MESSAGE
demo1-compact-backup Complete [READ_META(17/17),COMPACT_WORK(1291/1291)]
```

If the `STATUS` field displays `Complete`, the compact log backup process has finished successfully.

Backup CR examples

Back up data of all clusters

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
```

```
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    sendCredToTikv: false
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    sendCredToTikv: false
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-s3
```

```
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  # ...
  br:
    cluster: demo1
    sendCredToTikv: false
  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 those 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 follows:

  ```
  kubectl apply -f backup-scheduler-aws-s3.yaml
  ```

  The content of `backup-scheduler-aws-s3.yaml` is as follows:

  ```
  ---
  apiVersion: br.pingcap.com/v1alpha1
  kind: BackupSchedule
  metadata:
    name: demo1-backup-schedule-s3
    namespace: test1
  spec:
  ```

```
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      # 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 follows:

```
kubectl apply -f backup-scheduler-aws-s3.yaml
```

The content of `backup-scheduler-aws-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
```

```
    backupType: full
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: false
      # 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 follows:

```
kubectl apply -f backup-scheduler-aws-s3.yaml
```

The content of `backup-scheduler-aws-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-s3
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    serviceAccount: tidb-backup-manager
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: false
```

```
    # 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 preceding 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 this 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 `test1` namespace.

```
kubectl apply -f integrated-backup-schedule-s3.yaml
```

The content of `integrated-backup-schedule-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-s3
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      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
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder-log
```

In the preceding 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 test1 -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 test1
```

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 test1
```

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 Integrated management of scheduled snapshot backup, log backup, and compact log backup

To accelerate downstream recovery, you can enable `CompactBackup` CR in the `BackupSchedule` CR. This feature periodically compacts log backup files in remote storage. You must enable log backup before using log backup compaction. This section extends the configuration from the previous section.

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.

Create `BackupSchedule`

1. Create a `BackupSchedule` CR named `integrated-backup-schedule-s3` in the `test1` namespace.

```
kubectl apply -f integrated-backup-schedule-s3.yaml
```

The content of `integrated-backup-schedule-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-s3
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  compactInterval: "1h"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      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
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
      bucket: my-bucket
      prefix: my-folder-log
  compactBackupTemplate:
    br:
      cluster: demo1
      sendCredToTikv: true
    s3:
      provider: aws
      secretName: s3-secret
      region: us-west-1
```

```
    bucket: my-bucket
    prefix: my-folder-log
```

In the preceding example of `integrated-backup-schedule-s3.yaml`, the `backupSchedule` ↪ configuration is based on the previous section, with the following additions for `compactBackup`:

- Added the `BackupSchedule.spec.compactInterval` field to specify the interval for log backup compaction. It is recommended not to exceed the interval of scheduled snapshot backups and to keep it between one-half to one-third of the scheduled snapshot backup interval.
- Added the `BackupSchedule.spec.compactBackupTemplate` field. Ensure that the `BackupSchedule.spec.compactBackupTemplate.s3` configuration matches the `BackupSchedule.spec.logBackupTemplate.s3` configuration.

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 test1 -o wide
```

A compact log backup task is created together with `backupSchedule`. You can check the `CompactBackup` CR using the following command:

```
kubectl get cpbk -n test1
```

7.4.4.1.6 Delete the backup CR

If you no longer need the backup CR, refer to Delete the Backup CR.

7.4.4.1.7 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 `test1` 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 an S3-compatible storage to TiDB using BR, take the following steps to prepare the restore environment:

Note:

- BR uses a fixed ServiceAccount name that must be `tidb-backup-manager`.
- Starting from TiDB Operator v2, the `apiGroup` for resources such as `Backup` and `Restore` changes from `pingcap.com` to `br.pingcap.com`.

1. Save the following content as the `backup-rbac.yaml` file to create the required role-based access control (RBAC) resources:

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. Execute the following command to create the RBAC resources in the `test1` namespace:

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. Grant permissions to the remote storage for the `test1` namespace:

- If you are using Amazon S3 to back up your cluster, you can grant permissions in three methods. For more information, see AWS account permissions.
- If you are using other S3-compatible storage (such as Ceph and MinIO) to back up your cluster, you can grant permissions by using AccessKey and SecretKey.

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 follows:

```
kubectl apply -f restore-full-s3.yaml
```

The content of `restore-full-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test1
spec:
  br:
    cluster: demo2
    # 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 follows:

```
kubectl apply -f restore-full-s3.yaml
```

The content of `restore-full-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test1
  annotations:
    iam.amazonaws.com/role: arn:aws:iam::123456789012:role/user
spec:
  br:
    cluster: demo2
    sendCredToTikv: false
    # 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 follows:

```
kubectl apply -f restore-full-s3.yaml
```

The content of `restore-full-s3.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-s3
  namespace: test1
spec:
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo2
    sendCredToTikv: false
    # 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 test1 -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 `test1` 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

For more information, see Step 1: Prepare the restore environment.

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: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-s3
  namespace: test1
spec:
  restoreMode: pitr
  br:
    cluster: demo3
  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 test1
```

```
NAME                       COMPLETIONS  ...
restore-demo3-restore-s3 1/1            ...
```

You can also check the restoration status by using the following command:

```
kubectl get restore -n test1 -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.

7.4.5 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 GCS 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 TiB) 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 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

Note:

- BR uses a fixed ServiceAccount name that must be `tidb-backup-manager`.
- Starting from TiDB Operator v2, the `apiGroup` for resources such as `Backup` and `Restore` changes from `pingcap.com` to `br.pingcap.com`.

1. Save the following content as the `backup-rbac.yaml` file to create the required role-based access control (RBAC) resources:

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. Execute the following command to create the RBAC resources in the `test1` namespace:

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. Grant permissions to the remote storage for the `test1` namespace:

   Refer to GCS account permissions.

Snapshot backup

Create the `Backup` CR to back up cluster data to GCS as follows:

```
kubectl apply -f full-backup-gcs.yaml
```

The content of `full-backup-gcs.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-gcs
  namespace: test1
spec:
  # backupType: full
  br:
    cluster: demo1
    # 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:

- Starting from 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.

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 test1 -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 no longer supported in TiDB Operator v2. It is recommended to use the `logSubcommand` field to ensure clear and consistent configuration.

Start log backup

1. In the `test1` 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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    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 test1
```

```
NAME                                  COMPLETIONS  ...
backup-demo1-log-backup-gcs-log-start 1/1          ...
```

3. View the newly created `Backup` CR:

```
kubectl get backup -n test1
```

```
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 test1
```

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 test1
```

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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
```

```
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    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 test1
```

```
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 test1
```

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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: test1
```

```
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    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 test1
```

```
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 test1
```

Change the value of `logSubcommand` to `log-stop`, then save and exit the editor. The modified content is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-gcs
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    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 test1
```

```
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 test1
   ```

   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: br.pingcap.com/v1alpha1
   kind: Backup
   metadata:
     name: demo1-backup-gcs
     namespace: test1
   spec:
     backupMode: log
     logSubcommand: log-start/log-pause/log-stop
     br:
       cluster: demo1
       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 test1
```

```
NAME                                        COMPLETIONS  ...
...
backup-demo1-log-backup-gcs-log-truncate 1/1      ...
```

3. View the `Backup` CR information:

```
kubectl describe backup -n test1
```

```
...
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 test1 -o wide
```

```
NAME                  MODE      STATUS    ...  LOGTRUNCATEUNTIL
demo1-log-backup-gcs log       Stopped   ...  2022-10-10T15:21:00+08:00
```

Compact log backup

For TiDB v9.0.0 and later versions, you can use a `CompactBackup` CR to compact log backup data into SST format, accelerating downstream PITR (Point-in-time recovery).

This section explains how to compact log backup based on the log backup example from previous sections.

In the `test1` namespace, create a `CompactBackup` CR named `demo1-compact-backup`.

```
kubectl apply -f compact-backup-demo1.yaml
```

The content of `compact-backup-demo1.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: CompactBackup
metadata:
  name: demo1-compact-backup
  namespace: test1
spec:
  startTs: "***"
  endTs: "***"
  concurrency: 8
  maxRetryTimes: 2
```

```
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: my-log-backup-folder
```

The `startTs` and `endTs` fields specify the time range for the logs to be compacted by `demo1-compact-backup`. Any log that contains at least one data write within this time range will be included in the compaction process. As a result, the final compacted data might include data written outside this range.

The `gcs` settings should be the same as the storage settings of the log backup to be compacted. `CompactBackup` reads log files from the corresponding location and compacts them.

View the status of log backup compaction

After creating the `CompactBackup` CR, TiDB Operator automatically starts compacting the log backup. You can check the backup status using the following command:

```
kubectl get cpbk -n test1
```

From the output, you can find the status of the `CompactBackup` CR named `demo1-compact-backup`. An example output is as follows:

```
NAME                    STATUS                  PROGRESS
    ↪                                       MESSAGE
demo1-compact-backup Complete [READ_META(17/17),COMPACT_WORK(1291/1291)]
```

If the `STATUS` field displays `Complete`, the compact log backup process has finished successfully.

Backup CR examples

Back up data of all clusters

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  br:
    cluster: demo1
```

```
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  tableFilter:
```

```
  - "db1.table1"
  br:
    cluster: demo1
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-gcs
  namespace: test1
spec:
  # backupType: full
  tableFilter:
  - "db1.table1"
  - "db1.table2"
  br:
    cluster: demo1
  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 those of Prepare for an ad-hoc backup.

Perform a scheduled snapshot backup

1. Create a `BackupSchedule` CR to back up cluster data as follows:

```
kubectl apply -f backup-schedule-gcs.yaml
```

The content of `backup-schedule-gcs.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-gcs
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    # Clean outdated backup data based on maxBackups or maxReservedTime
      ↪ . If not configured, the default policy is Retain
    # cleanPolicy: Delete
    br:
      cluster: demo1
      # 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 preceding `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 test1 -o wide
```

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
```

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 `test1` namespace.

```
kubectl apply -f integrated-backup-scheduler-gcs.yaml
```

The content of `integrated-backup-scheduler-gcs.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-gcs
  namespace: test1
spec:
```

```
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-snapshot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    gcs:
      projectId: ${project_id}
      secretName: gcs-secret
      bucket: my-bucket
      prefix: schedule-backup-folder-log
```

In the preceding 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 test1 -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 test1
```

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 test1
```

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 Integrated management of scheduled snapshot backup, log backup, and compact log backup

To accelerate downstream recovery, you can enable `CompactBackup` CR in the `BackupSchedule` CR. This feature periodically compacts log backup files in remote storage. You must enable log backup before using log backup compaction. This section extends the configuration from the previous section.

Prerequisites: Prepare for a scheduled snapshot backup

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 `test1` namespace.

```
kubectl apply -f integrated-backup-scheduler-gcs.yaml
```

The content of `integrated-backup-scheduler-gcs.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-gcs
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  compactInterval: "1h"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
```

```
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: schedule-backup-folder-snapshot
logBackupTemplate:
  backupMode: log
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: schedule-backup-folder-log
compactBackupTemplate:
  br:
    cluster: demo1
    sendCredToTikv: true
  gcs:
    projectId: ${project_id}
    secretName: gcs-secret
    bucket: my-bucket
    prefix: schedule-backup-folder-log
```

In the preceding example of `integrated-backup-schedule-gcs.yaml`, the `backupSchedule` ↪ configuration is based on the previous section, with the following additions for `compactBackup`:

- Added the `BackupSchedule.spec.compactInterval` field to specify the time interval for log backup compaction. It is recommended not to exceed the interval of scheduled snapshot backups and to keep it between one-half to one-third of the scheduled snapshot backup interval.
- Added the `BackupSchedule.spec.compactBackupTemplate` field. Ensure that the `BackupSchedule.spec.compactBackupTemplate.gcs` configuration matches the `BackupSchedule.spec.logBackupTemplate.gcs` configuration.

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 test1 -o wide
```

A compact log backup task is created together with `backupSchedule`. You can check the `CompactBackup` CR using the following command:

```
kubectl get cpbk -n test1
```

7.4.5.1.6 Delete the backup CR

If you no longer need the backup CR, refer to Delete the Backup CR.

7.4.5.1.7 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 Custom Resource Definition (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.

7.4.5.2.1 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 `test1` 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:

> **Note:**
>
> - BR uses a fixed ServiceAccount name that must be `tidb-backup-manager`.
> - Starting from TiDB Operator v2, the `apiGroup` for resources such as `Backup` and `Restore` changes from `pingcap.com` to `br.pingcap.com`.

1. Save the following content as the `backup-rbac.yaml` file to create the required role-based access control (RBAC) resources:

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]

---
kind: ServiceAccount
apiVersion: v1
```

```
metadata:
  name: tidb-backup-manager

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
```

2. Execute the following command to create the RBAC resources in the `test1` namespace:

```
kubectl apply -f backup-rbac.yaml -n test1
```

3. Grant permissions to the remote storage for the `test1` namespace:

   Refer to GCS account permissions.

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: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-gcs
  namespace: test1
spec:
  # backupType: full
  br:
    cluster: demo2
    # 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 test1 -o wide
```

```
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 `test1` 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

Refer to Step 1: Prepare the restore environment.

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 `test1` 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: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-gcs
  namespace: test1
spec:
  restoreMode: pitr
  br:
    cluster: demo3
  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 test1
```

```
NAME                       COMPLETIONS  ...
restore-demo3-restore-gcs  1/1          ...
```

You can also check the restoration status by using the following command:

```
kubectl get restore -n test1 -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.

7.4.6 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 TiB) 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 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 the required role-based access control (RBAC) resources:

```
kubectl apply -n test1 -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]
---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
EOF
```

2. Refer to Grant permissions to an Azure account to grant access to remote storage. Azure provides two methods for granting permissions. After successful authorization,

a Secret object named `azblob-secret` or `azblob-secret-ad` should exist in the `test1` namespace.

Note:

- The authorized account must have at least write access to Blob data, such as the Contributor role.
- When creating the Secret object, you can customize its name. For demonstration purposes, this document uses `azblob-secret` as the example Secret name.

Snapshot backup

To perform a snapshot backup, take the following steps:

Create the `Backup` CR named `demo1-full-backup-azblob` in the `test1` namespace:

```
kubectl apply -f full-backup-azblob.yaml
```

The content of `full-backup-azblob.yaml` is as follows:

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-full-backup-azblob
  namespace: test1
spec:
  backupType: full
  br:
    cluster: demo1
    # 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 configuring the `full-backup-azblob.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.
- 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 you specified when creating the Secret object, such as `azblob-secret`.
- 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 test1 -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.

Start log backup

1. In the `test1` 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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    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 test1
```

```
NAME                                      COMPLETIONS  ...
backup-demo1-log-backup-azblob-log-start 1/1          ...
```

3. View the newly created `Backup` CR:

```
kubectl get backup -n test1
```

```
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 test1
```

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 test1
```

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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-pause
  br:
    cluster: demo1
    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 test1
```

```
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 test1
```

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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start
  br:
    cluster: demo1
    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 `Pause` to `Running`:

```
kubectl get backup -n test1
```

```
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 test1
```

Change the value of `logSubcommand` to `log-stop`, then save and exit the editor. The modified content is as follows:

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-stop
  br:
    cluster: demo1
    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 test1
```

```
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.

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 test1
```

   In the last line of the CR, append `spec.logTruncateUntil: "2022-10-10T15 ↪ :21:00+08:00"`. Then save and exit the editor. The modified content is as follows:

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-log-backup-azblob
  namespace: test1
spec:
  backupMode: log
  logSubcommand: log-start/log-pause/log-stop
  br:
    cluster: demo1
    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 test1
```

```
NAME                                          COMPLETIONS  ...
...
backup-demo1-log-backup-azblob-log-truncate 1/1      ...
```

3. View the `Backup` CR information:

```
kubectl describe backup -n test1
```

```
...
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 test1 -o wide
```

```
NAME                MODE      STATUS    ...   LOGTRUNCATEUNTIL
demo1-log-backup-azblob log       Complete  ...   2022-10-10T15
    ↪ :21:00+08:00
```

Compact log backup

For TiDB v9.0.0 and later versions, you can use a `CompactBackup` CR to compact log backup data into SST format, accelerating downstream PITR (Point-in-time recovery).

This section explains how to compact log backup based on the log backup example from previous sections.

In the `test1` namespace, create a `CompactBackup` CR named `demo1-compact-backup`.

```
kubectl apply -f compact-backup-demo1.yaml
```

The content of `compact-backup-demo1.yaml` is as follows:

```
apiVersion: br.pingcap.com/v1alpha1
kind: CompactBackup
metadata:
  name: demo1-compact-backup
  namespace: test1
spec:
  startTs: "***"
  endTs: "***"
  concurrency: 8
  maxRetryTimes: 2
  br:
    cluster: demo1
    sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-log-backup-folder
```

The `startTs` and `endTs` fields specify the time range for the logs to be compacted by `demo1-compact-backup`. Any log that contains at least one write within this time range will be included in the compaction process. As a result, the final compacted data might include data written outside this range.

The `azblob` settings should be the same as the storage settings of the log backup to be compacted. `CompactBackup` reads log files from the corresponding location and compacts them.

View the status of log backup compaction

After creating the `CompactBackup` CR, TiDB Operator automatically starts compacting the log backup. You can check the backup status using the following command:

```
kubectl get cpbk -n test1
```

From the output, you can find the status of the `CompactBackup` CR named `demo1-compact-backup`. An example output is as follows:

```
NAME                  STATUS                  PROGRESS
    ↪                                      MESSAGE
demo1-compact-backup Complete [READ_META(17/17),COMPACT_WORK(1291/1291)]
```

If the `STATUS` field displays `Complete`, the compact log backup process has finished successfully.

Backup CR examples

Back up data of all clusters

```
apiVersion: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  br:
    cluster: demo1
    sendCredToTikv: false
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
```

```
  tableFilter:
  - "db1.*"
  br:
    cluster: demo1
    sendCredToTikv: false
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  br:
    cluster: demo1
    sendCredToTikv: false
  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: br.pingcap.com/v1alpha1
kind: Backup
metadata:
  name: demo1-backup-azblob
  namespace: test1
spec:
  backupType: full
  serviceAccount: tidb-backup-manager
  tableFilter:
  - "db1.table1"
  - "db1.table2"
```

```
# ...
br:
  cluster: demo1
  sendCredToTikv: false
azblob:
  secretName: azblob-secret-ad
  container: my-container
  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 follows:

```
kubectl apply -f backup-scheduler-azblob.yaml
```

The content of `backup-scheduler-azblob.yaml` is as follows:

```
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-azblob
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    br:
      cluster: demo1
      # 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 follows:

```
kubectl apply -f backup-scheduler-azblob.yaml
```

The content of `backup-scheduler-azblob.yaml` is as follows:

```
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: demo1-backup-schedule-azblob
  namespace: test1
spec:
  #maxBackups: 5
  #pause: true
  maxReservedTime: "3h"
  schedule: "*/2 * * * *"
  backupTemplate:
    backupType: full
    br:
      cluster: demo1
      sendCredToTikv: false
      # 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 test1 -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 test1
```

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 `test1` namespace.

   ```
   kubectl apply -f integrated-backup-scheduler-azblob.yaml
   ```

   The content of `integrated-backup-scheduler-azblob.yaml` is as follows:

   ```
   apiVersion: br.pingcap.com/v1alpha1
   kind: BackupSchedule
   metadata:
     name: integrated-backup-schedule-azblob
     namespace: test1
   spec:
     maxReservedTime: "3h"
   ```

```
  schedule: "* */2 * * *"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: schedule-backup-folder-snapshot
      #accessTier: Hot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: schedule-backup-folder-log
      #accessTier: Hot
```

In the preceding 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 test1 -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 test1
```

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 test1
```

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 Integrated management of scheduled snapshot backup, log backup, and compact log backup

To accelerate downstream recovery, you can enable `CompactBackup` CR in the `BackupSchedule` CR. This feature periodically compacts log backup files in remote storage. You must enable log backup before using log backup compaction. This section extends the configuration from the previous section.

Prerequisites: Prepare for a scheduled snapshot backup

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 `test1` namespace.

```
kubectl apply -f integrated-backup-scheduler-azblob.yaml
```

The content of `integrated-backup-scheduler-azblob.yaml` is as follows:

```
apiVersion: br.pingcap.com/v1alpha1
kind: BackupSchedule
metadata:
  name: integrated-backup-schedule-azblob
  namespace: test1
spec:
  maxReservedTime: "3h"
  schedule: "* */2 * * *"
  compactInterval: "1h"
  backupTemplate:
    backupType: full
    cleanPolicy: Delete
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
```

```
    container: my-container
    prefix: schedule-backup-folder-snapshot
    #accessTier: Hot
  logBackupTemplate:
    backupMode: log
    br:
      cluster: demo1
      sendCredToTikv: true
    azblob:
      secretName: azblob-secret
      container: my-container
      prefix: schedule-backup-folder-log
      #accessTier: Hot
```

In the preceding example of `integrated-backup-schedule-azblob.yaml`, the `backupSchedule` configuration is based on the previous section, with the following additions for `compactBackup`:

- Added the `BackupSchedule.spec.compactInterval` field to specify the time interval for log backup compaction. It is recommended not to exceed the interval of scheduled snapshot backups and to keep it between one-half to one-third of the scheduled snapshot backup interval.
- Added the `BackupSchedule.spec.compactBackupTemplate` field. Ensure that the `BackupSchedule.spec.compactBackupTemplate.azblob` configuration matches the `BackupSchedule.spec.logBackupTemplate.azblob` configuration.

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 test1 -o wide
```

A compact log backup task is created together with `backupSchedule`. You can check the `CompactBackup` CR using the following command:

```
kubectl get cpbk -n test1
```

7.4.6.1.6 Delete the backup CR

If you no longer need the backup CR, you can delete it by referring to Delete the Backup CR.

7.4.6.1.7 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 Custom Resource Definition (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 enables 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 the required role-based access control (RBAC) resources:

```
kubectl apply -n test2 -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
rules:
- apiGroups: [""]
  resources: ["events"]
  verbs: ["*"]
- apiGroups: ["br.pingcap.com"]
  resources: ["backups", "restores"]
  verbs: ["get", "watch", "list", "update"]
---
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tidb-backup-manager
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: tidb-backup-manager
  labels:
    app.kubernetes.io/component: tidb-backup-manager
subjects:
- kind: ServiceAccount
  name: tidb-backup-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: tidb-backup-manager
EOF
```

2. Refer to Grant permissions to an Azure account to grant access to remote storage. Azure provides two methods for granting permissions. After successful authorization, a Secret object named `azblob-secret` or `azblob-secret-ad` should exist in the namespace.

Note:

- The authorized account must have at least write access to Blob data, such as the Contributor role.
- When creating the Secret object, you can customize its name. For demonstration purposes, this document uses `azblob-secret` as the example Secret name.

Step 2: Restore the backup data to a TiDB cluster

Create a `Restore` CR named `demo2-restore-azblob` in the `test2` namespace to restore cluster data as follows:

```
kubectl apply -n test2 -f restore-full-azblob.yaml
```

The content of `restore-full-azblob.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo2-restore-azblob
  namespace: test2
spec:
  br:
    cluster: demo2
    # logLevel: info
    # statusAddr: ${status_addr}
    # concurrency: 4
    # rateLimit: 0
    # timeAgo: ${time}
    # checksum: true
    # sendCredToTikv: true
  azblob:
    secretName: azblob-secret
    container: my-container
    prefix: my-full-backup-folder
```

When configuring `restore-full-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 you specified when creating the Secret object, such as `azblob-secret`.
- 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 test2 -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

The steps to prepare for a PITR are the same as those of Full restoration.

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 `test3` namespace and specify the restoration time point as `2022-10-10T17:21:00+08:00`:

```
kubectl apply -n test3 -f restore-point-azblob.yaml
```

The content of `restore-point-azblob.yaml` is as follows:

```
---
apiVersion: br.pingcap.com/v1alpha1
kind: Restore
metadata:
  name: demo3-restore-azblob
  namespace: test3
spec:
  restoreMode: pitr
  br:
    cluster: demo3
  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 test3
```

```
NAME                          COMPLETIONS  ...
restore-demo3-restore-azblob 1/1          ...
```

You can also check the restoration status by using the following command:

```
kubectl get restore -n test3 -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.

7.5 Maintain

7.5.1 View TiDB Logs on Kubernetes

This document introduces the methods to view logs of TiDB components and TiDB slow log.

7.5.1.1 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`.

7.5.1.2 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.

7.5.2 Pause Sync of a TiDB Cluster on Kubernetes

This document introduces how to pause sync of a TiDB cluster on Kubernetes by modifying its configuration.

7.5.2.1 What is sync in TiDB Operator

In TiDB Operator, the controller constantly compares the desired state recorded in the Custom Resource (CR) object with the actual state. The controller then creates, updates, or deletes Kubernetes resources to ensure the TiDB cluster matches the desired state. This ongoing adjustment process is referred to as **sync**. For more information, see TiDB Operator Architecture.

7.5.2.2 Usage scenarios

The following lists 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.

7.5.2.3 Pause TiDB cluster sync

To pause sync of a TiDB cluster, set `spec.paused: true` in the Cluster CR.

1. Run the following command to modify the TiDB cluster configuration. `${cluster_name}` represents the name of the TiDB cluster, and `${namespace}` represents the TiDB cluster namespace.

```
kubectl patch cluster ${cluster_name} -n ${namespace} --type merge -p
    ↪ '{"spec":{"paused": true}}'
```

2. After the sync is paused, you can confirm the cluster status by checking the TiDB Operator Pod logs. `${pod_name}` represents the name of TiDB Operator Pod, and `${namespace}` represents the namespace of TiDB Operator.

```
kubectl logs ${pod_name} -n ${namespace} | grep paused
```

The following output indicates that the sync of all components in the TiDB cluster is paused.

```
```
2025-04-25T09:27:27.866Z INFO   TiCDC   cluster paused is updating {"from":
    ↪ false, "to": true}
2025-04-25T09:27:27.866Z INFO   TiDB   cluster paused is updating {"from":
    ↪ false, "to": true}
2025-04-25T09:27:27.867Z INFO   TiFlash cluster paused is updating {"from":
    ↪ false, "to": true}
2025-04-25T09:27:27.868Z INFO   PD   cluster paused is updating {"from":
    ↪ false, "to": true}
2025-04-25T09:27:27.868Z INFO   TiKV   cluster paused is updating {"from":
    ↪ false, "to": true}
```
```

7.5.2.4 Resume TiDB cluster sync

To resume the sync of the TiDB cluster, set `spec.paused: false` in the Cluster CR. Once resumed, TiDB Operator immediately processes all configuration changes that accumulated during the pause.

1. Run the following command to modify the TiDB cluster configuration. `${cluster_name}` represents the name of the TiDB cluster, and `${namespace}` represents the TiDB cluster namespace.

```
kubectl patch cluster ${cluster_name} -n ${namespace} --type merge -p
    ↪ '{"spec":{"paused": false}}'
```

2. After the sync is resumed, you can confirm the cluster status by checking the TiDB Operator Pod logs. `${pod_name}` represents the name of TiDB Operator Pod, and `${namespace}` represents the namespace of TiDB Operator.

```
kubectl logs ${pod_name} -n ${namespace} | grep "paused"
```

The following output shows that the timestamp when the `cluster paused` status changes from `true` to `false` is later than the timestamp when it changes from `false` to `true`, indicating that the sync is resumed.

```
2025-04-25T09:32:38.867Z INFO  TiKV    cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO  PD      cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO  TiFlash cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO  TiCDC   cluster paused is updating {"
    ↪ from": true, "to": false}
2025-04-25T09:32:38.868Z INFO  TiDB    cluster paused is updating {"
    ↪ from": true, "to": false}
```

7.5.3 Suspend and Resume a TiDB Cluster on Kubernetes

This document describes how to suspend and resume a TiDB cluster on Kubernetes by configuring the `Cluster` object. When you suspend a cluster, all component Pods are stopped, but the `Cluster` object and associated resources (such as Services and PVCs) are retained. This preserves the cluster's data and configuration for later recovery.

7.5.3.1 Usage scenarios

You can suspend a TiDB cluster in the following scenarios:

- Temporarily release compute resources in a test environment.
- Stop a development cluster that is unused for an extended period.
- Pause a cluster temporarily while retaining its data and configuration.

7.5.3.2 Before you begin

Before you suspend a TiDB cluster, consider the following:

- Suspending the cluster interrupts cluster services.

- Existing client connections are terminated forcefully.
- PVCs and data are retained and continue to occupy storage space.
- Services and configurations associated with the cluster remain unchanged.

7.5.3.3 Suspend a TiDB cluster

To suspend a TiDB cluster, perform the following steps:

1. In the `Cluster` object, set the `spec.suspendAction.suspendCompute` field to `true` to suspend the entire TiDB cluster:

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  suspendAction:
    suspendCompute: true
  # ...
```

2. After the cluster is suspended, run the following command to observe the Pods being gradually deleted:

```
kubectl -n ${namespace} get pods -w
```

7.5.3.4 Resume a TiDB cluster

To resume a suspended TiDB cluster, perform the following steps:

1. In the `Cluster` object, set the `spec.suspendAction.suspendCompute` field to `false` to resume the suspended TiDB cluster:

```
apiVersion: core.pingcap.com/v1alpha1
kind: Cluster
metadata:
  name: ${cluster_name}
  namespace: ${namespace}
spec:
  suspendAction:
    suspendCompute: false
  # ...
```

2. After the cluster is resumed, run the following command to observe the Pods being gradually created:

```
kubectl -n ${namespace} get pods -w
```

7.5.4 Restart a TiDB Cluster on Kubernetes

When using a TiDB cluster, you might need to restart it if a Pod encounters issues such as memory leaks. This document describes how to perform a graceful rolling restart of all Pods in a component or a graceful restart of a specific Pod.

> **Warning:**
>
> In production environments, it is strongly recommended not to forcefully delete Pods in the TiDB cluster. Although TiDB Operator will automatically recreate deleted Pods, this could cause some requests to the TiDB cluster to fail.

7.5.4.1 Perform a graceful rolling restart of all Pods in a component

To perform a graceful rolling restart of all Pods in a component (such as PD, TiKV, or TiDB), modify the corresponding Component Group Custom Resource (CR) configuration by adding a `pingcap.com/restartedAt` label or annotation under the `.spec.template.metadata` section and setting its value to a string that ensures idempotency, such as a timestamp.

The following example shows how to add an annotation for the PD component to trigger a graceful rolling restart of all PD Pods in the `PDGroup`:

```
apiVersion: core.pingcap.com/v1alpha1
kind: PDGroup
metadata:
  name: pd
spec:
  replicas: 3
  template:
    metadata:
      annotations:
        pingcap.com/restartedAt: 2025-06-30T12:00
```

7.5.4.2 Perform a graceful restart of a single Pod in a component

You can restart a specific Pod in the TiDB cluster. The process differs slightly depending on the component.

For a TiKV Pod, specify the `--grace-period` option when deleting the Pod to provide sufficient time to evict the Region leader. Otherwise, the operation might fail. The following command sets a 60-second grace period for the TiKV Pod:

```
kubectl -n ${namespace} delete pod ${pod_name} --grace-period=60
```

For other component Pods, you can delete them directly, because TiDB Operator will automatically handle a graceful restart:

```
kubectl -n ${namespace} delete pod ${pod_name}
```

7.5.5 Destroy TiDB Clusters on Kubernetes

This document describes how to destroy TiDB clusters on Kubernetes.

7.5.5.1 Destroy a TiDB cluster managed by `Cluster`

To destroy a TiDB cluster managed by `Cluster`, run the following command:

```
kubectl delete cluster ${cluster_name} -n ${namespace}
```

8 Reference

8.1 Architecture

8.1.1 TiDB Operator Architecture

This document introduces the architecture of TiDB Operator and how it works.

8.1.1.1 Architecture

The following diagram shows an overview of the TiDB Operator architecture:

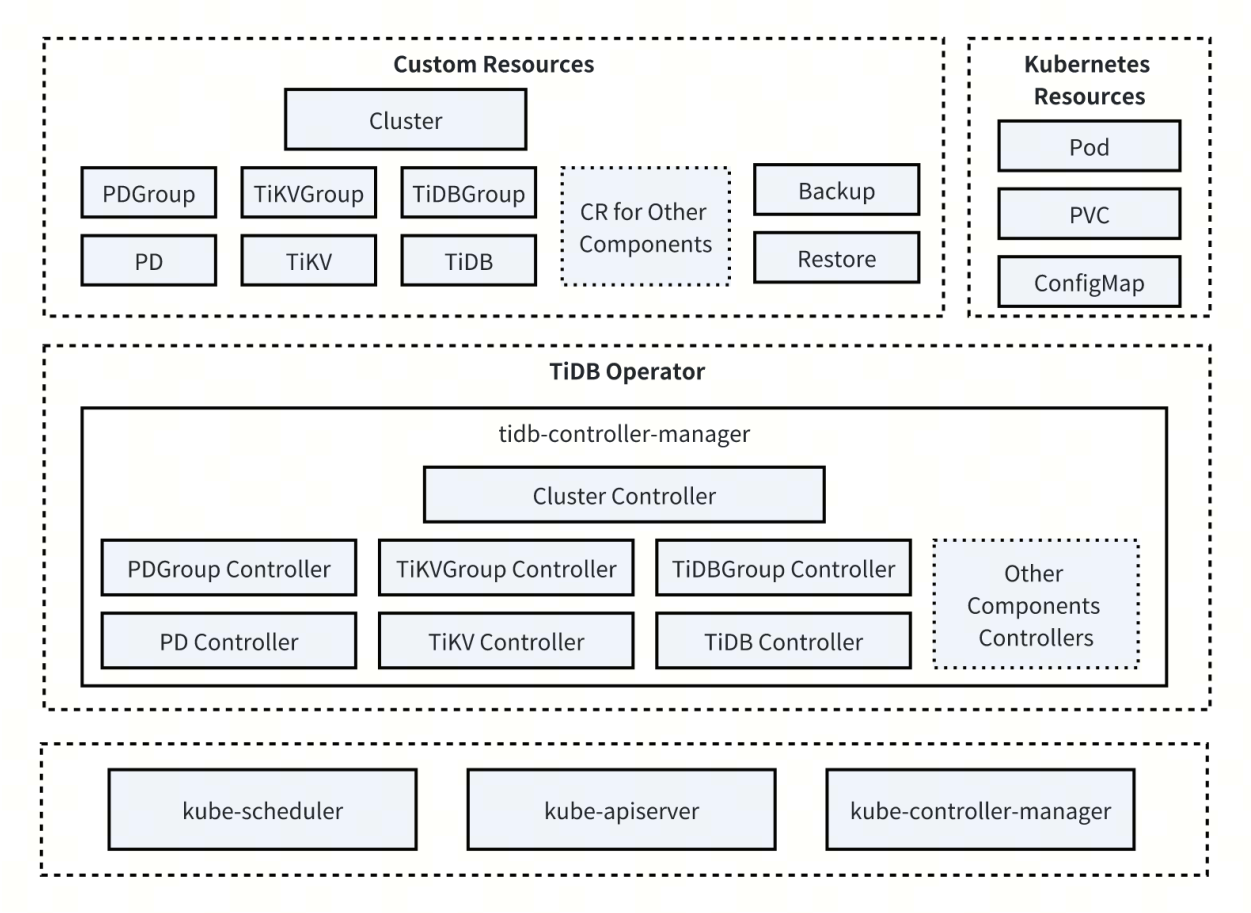

Figure 2: TiDB Operator Architecture

The diagram includes several resource objects defined by Custom Resource Definitions (CRDs), such as `Cluster`, `PDGroup`, `PD`, `TiKVGroup`, `TiKV`, `TiDBGroup`, `TiDB`, `Backup`, and `Restore`. The following describes some of these resources:

- `Cluster`: represents a complete TiDB cluster. It contains shared configuration and feature flags for the TiDB cluster and reflects the overall cluster status. This CRD is designed as the "namespace" for the TiDB cluster, and all components in the cluster must reference a `Cluster` CR.
- `ComponentGroup`: describes a group of TiDB cluster components with the same configuration. For example:
  - `PDGroup`: a group of PD instances with the same configuration.
  - `TiKVGroup`: a group of TiKV instances with the same configuration.
  - `TiDBGroup`: a group of TiDB instances with the same configuration.
- `Component`: describes an individual TiDB cluster component. For example:

  - `PD`: a single PD instance.
  - `TiKV`: a single TiKV instance.
  - `TiDB`: a single TiDB instance.

- `Backup`: describes a backup task for the TiDB cluster.
- `Restore`: describes a restore task for the TiDB cluster.

8.1.1.2 Control flow

TiDB Operator uses a declarative API to automate cluster management by continuously monitoring user-defined resources. The core workflow is as follows:

1. The user creates `Cluster` and other component Custom Resource (CR) objects through `kubectl`, such as `PDGroup`, `TiKVGroup`, and `TiDBGroup`.
2. TiDB Operator continuously watches these CRs and dynamically adjusts the corresponding `Pod`, `PVC`, and `ConfigMap` objects based on the actual cluster state.

Through this control (reconciliation) loop, TiDB Operator can automatically perform cluster node health checks and failure recovery. Operations such as deployment, upgrades, and scaling can also be completed with one action by modifying the `Cluster` and other component CRs.

The following diagram shows the control flow using TiKV as an example:

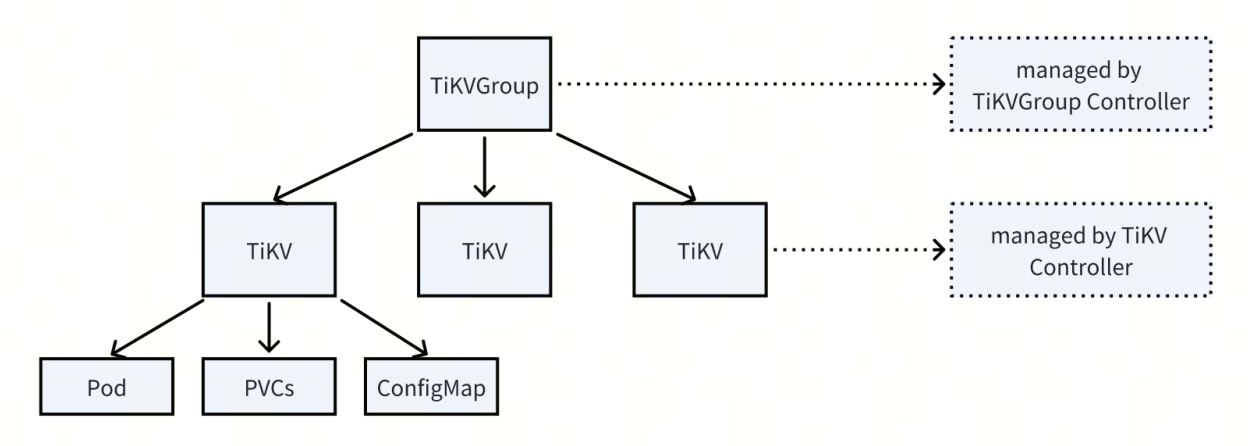

Figure 3: TiDB Operator Control Flow

In this workflow:

- TiKVGroup Controller: watches the `TiKVGroup` CR and creates or updates the corresponding `TiKV` CR based on its configuration.
- TiKV Controller: watches the `TiKV` CR and creates or updates related resources such as Pod, PVC, and ConfigMap based on the configuration in the CR.

8.2 Comparison Between TiDB Operator v2 and v1

With the rapid development of TiDB and the Kubernetes ecosystem, the existing architecture and implementation of TiDB Operator v1 have encountered some challenges. To better adapt to these changes, TiDB Operator v2 introduces a major refactor of v1.

8.2.1 Core changes in TiDB Operator v2

8.2.1.1 Split the `TidbCluster` CRD

Initially, the TiDB cluster has only three core components: PD, TiKV, and TiDB. To simplify deployment and reduce user cognitive load, all components of the TiDB cluster are defined in a single CRD, `TidbCluster`. However, as TiDB evolves, this design faces several challenges:

- The number of TiDB cluster components has increased, with eight components currently defined in the `TidbCluster` CRD.
- To display status, the state of all nodes is defined in the `TidbCluster` CRD.
- Heterogeneous clusters are not considered initially, so additional `TidbCluster` CRs has to be introduced to support them.
- The `/scale` API is not supported, making it impossible to integrate with Kubernetes HorizontalPodAutoscaler (HPA).
- A large CR/CRD can cause difficult-to-resolve performance issues.

TiDB Operator v2 addresses these issues by splitting `TidbCluster` into multiple independent CRDs by component.

8.2.1.2 Remove the StatefulSet dependency and manage Pods directly

Due to the complexity of TiDB clusters, Kubernetes' native deployment and StatefulSet controllers cannot fully meet TiDB's deployment and operation needs. TiDB Operator v1 manages all TiDB components using StatefulSet, but some limitations of StatefulSet prevent maximizing Kubernetes' capabilities, such as:

- StatefulSet restricts modifications to `VolumeClaimTemplate`, making native scaling impossible.
- StatefulSet enforces the order of scaling and rolling updates, causing repeated leader scheduling.
- StatefulSet requires all Pods under the same controller to have identical configurations, necessitating complex startup scripts to differentiate Pod parameters.
- There is no API for defining raft members, leading to semantic conflicts between restarting Pods and removing raft members, and no intuitive way to remove a specific TiKV node.

TiDB Operator v2 removes the dependency on StatefulSet and introduces the following CRDs:

- `Cluster`
- `ComponentGroup`
- `Instance`

These three-layer CRDs can manage Pods directly. TiDB Operator v2 uses the `ComponentGroup` CRD to manage nodes with common characteristics, reducing complexity, and the `Instance` CRD to facilitate management of individual stateful instances, providing instance-level operations and ensuring flexibility.

Benefits include:

- Better support for volume configuration changes.
- More reasonable rolling update order, such as restarting the leader last to prevent repeated leader migration.
- In-place upgrades for non-core components (such as log tail and istio), reducing the impact of TiDB Operator and infrastructure changes on the TiDB cluster.
- Graceful Pod restarts using `kubectl delete ${pod}` and rebuilding specific TiKV nodes using `kubectl delete ${instance}`.
- More intuitive status display.

8.2.1.3 Introduce the Overlay mechanism and no longer manage Kubernetes fields unrelated to TiDB directly

Each new version of Kubernetes may introduce new fields that users need, but TiDB Operator may not care about these fields. In v1, a lot of development effort was spent supporting new Kubernetes features, including manually adding new fields to the `TidbCluster` CRD and propagating them. TiDB Operator v2 introduces the Overlay mechanism, which supports all new Kubernetes resource fields (especially for Pods) in a unified way. For details, see Overlay.

8.2.1.4 Other new features in TiDB Operator v2

8.2.1.4.1 Enhance validation capabilities

TiDB Operator v2 enhances configuration validation through Validation Rules and Validating Admission Policies, improving usability and robustness.

8.2.1.4.2 Support `/status` and `/scale` sub resources

TiDB Operator v2 supports CRD sub resources and can integrate with Kubernetes HPA for automated scaling.

8.2.1.4.3 Remove `tidb scheduler` component and support the evenly spread policy

TiDB Operator v2 supports configuring the evenly spread policy to distribute components evenly across regions and zones as needed, and removes the `tidb scheduler` component.

8.2.2 Components and features not yet supported in TiDB Operator v2

8.2.2.1 Components

8.2.2.1.1 `Binlog` (Pump + Drainer)

This component is deprecated. For more information, see TiDB Binlog Overview.

8.2.2.1.2 Dumpling + TiDB Lightning

TiDB Operator no longer provides direct support. It is recommended to use native Kubernetes jobs to run them.

8.2.2.1.3 `TidbInitializer`

TiDB Operator v2 no longer supports this CRD. You can use BootstrapSQL to run initialization SQL statements.

8.2.2.1.4 `TidbMonitor`

TiDB Operator v2 no longer supports this CRD. Because monitoring systems are often complex and varied, `TidbMonitor` is difficult to integrate into production-grade monitoring systems. TiDB provides more flexible solutions for integrating with common monitoring systems, rather than running a Prometheus + Grafana + Alert-Manager combination through CRD. For details, see Deploy Monitoring and Alerts for a TiDB Cluster.

8.2.2.1.5 `TidbNgMonitoring`

Not supported yet.

8.2.2.1.6 `TidbDashboard`

Deployment through CRD is not supported. You can use the built-in dashboard or deploy it yourself through Deployment.

8.2.2.2 Features

8.2.2.2.1 Cross-namespace deployment

Not supported due to potential security issues and unclear user scenarios.

8.2.2.2.2 Cross-Kubernetes cluster deployment

Not supported due to potential security issues and unclear user scenarios.

8.2.2.2.3 Back up and restore based on EBS volume snapshots

Backup based on EBS volume snapshots has the following unsolvable issues:

- High cost. EBS volume snapshots are very expensive.
- Long RTO. Recovery from EBS volume snapshots takes a long time.

With continuous optimization, TiDB BR performance has greatly improved, so backup and restore based on EBS volume snapshots is no longer necessary. Therefore, TiDB Operator v2 no longer supports this feature.

8.3 Tools

8.3.1 Tools on Kubernetes

This document describes how to use operational tools for TiDB on Kubernetes, including PD Control, TiKV Control, and TiDB Control.

8.3.1.1 Use PD Control on Kubernetes

PD Control is the command-line tool for PD (Placement Driver). To use PD Control to operate TiDB clusters on Kubernetes, first establish a local connection to the PD service using `kubectl port-forward`:

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd 2379:2379 &>/
    ↪ tmp/portforward-pd.log &
```

After running this command, you can access the PD service at `127.0.0.1:2379` and use the default parameters of the `pd-ctl` command directly. For example, to view the PD configuration:

```
pd-ctl -d config show
```

If port 2379 is already in use, you can specify another local port:

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd ${local_port
    ↪ }:2379 &>/tmp/portforward-pd.log &
```

In this case, you need to explicitly specify the PD port in the `pd-ctl` command:

```
pd-ctl -u 127.0.0.1:${local_port} -d config show
```

8.3.1.2 Use TiKV Control on Kubernetes

TiKV Control is the command-line tool for TiKV. To use TiKV Control to operate TiDB clusters on Kubernetes, first establish local connections to the PD service and the target TiKV Pod using `kubectl port-forward`:

The following example forwards the local port 2379 to the PD service:

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd 2379:2379 &>/
    ↪ tmp/portforward-pd.log &
```

The following example forwards the local port 20160 to the TiKV Pod:

```
kubectl port-forward -n ${namespace} ${pod_name} 20160:20160 &>/tmp/
    ↪ portforward-tikv.log &
```

After the connections are established, you can access the PD service and TiKV node through the corresponding local ports:

```
tikv-ctl --host 127.0.0.1:20160 --pd 127.0.0.1:2379 ${subcommands}
```

8.3.1.3 Use TiDB Control on Kubernetes

TiDB Control is the command-line tool for TiDB. To use TiDB Control, you need local access to both the TiDB node and the PD service. It is recommended to use `kubectl port-forward` to establish these connections.

The following example forwards the local port 2379 to the PD service:

```
kubectl port-forward -n ${namespace} svc/${pd_group_name}-pd 2379:2379 &>/
    ↪ tmp/portforward-pd.log &
```

The following example forwards the local port 10080 to the TiDB Pod:

```
kubectl port-forward -n ${namespace} ${pod_name} 10080:10080 &>/tmp/
    ↪ portforward-tidb.log &
```

After the connections are established, you can run `tidb-ctl` to perform various operations. For example, to view the schema of the `mysql` database:

```
tidb-ctl schema in mysql
```

9 Release Notes

9.1 v2.0

9.1.1 TiDB Operator 2.0.0-beta.0 Release Notes

Release date: July 9, 2025

TiDB Operator version: 2.0.0-beta.0

With the rapid development of TiDB and the Kubernetes ecosystem, TiDB Operator releases v2.0.0-beta.0, which includes comprehensive refactoring from v1 to provide a more stable, efficient, and maintainable cluster management experience.

For a detailed comparison between v2 and v1, see Comparison Between TiDB Operator v2 and v1.

> **Warning:**
>
> This is a beta release. **Test thoroughly before deploying in production environments.**

9.1.1.1 Major changes and improvements

9.1.1.1.1 Core architecture refactoring

TiDB Operator v2 includes a comprehensive redesign of the v1 architecture, with the following key changes:

- **CRD splitting**: split the v1 `TidbCluster` CRD into multiple independent CRDs for more granular component management, improving maintainability and flexibility.
- **Direct Pod management**: remove the dependency on StatefulSet. Pods are now managed directly, providing higher flexibility and more precise control over Pod lifecycle and scheduling behavior.
- **Controller architecture upgrade**: implement controller logic based on the controller-runtime framework. This simplifies controller development, improves development efficiency, and enhances system stability and reliability.

9.1.1.1.2 New features and enhancements

- **Support the Overlay field**:
  - Enable you to flexibly specify all Kubernetes-supported fields for Pods without modifying the TiDB Operator source code.
  - Provide security validation mechanisms to prevent accidental overwrites of critical system labels.
- **Support topology-aware scheduling**:
  - Support the `EvenlySpread` strategy to evenly distribute Pods across topology domains.

  - Support topology weight configuration for flexible control of instance distribution ratios across topology domains.
  - Enhance cluster high availability and fault tolerance.

- **Enhance the field validation**:
  - Integrate Kubernetes Validation Rules and Validating Admission Policy.
  - Support field format and value range validation.
  - Provide clear and user-friendly error messages to facilitate troubleshooting.

- **Support CRD subresources**:
  - Support the `status` subresource for unified status management.
  - Support the `scale` subresource to integrate with HorizontalPodAutoscaler (HPA), enabling auto-scaling.
  - Improve compatibility with the Kubernetes ecosystem.

- **Optimize the configuration management**:
  - Optimize the configuration hash algorithm to avoid unnecessary rolling updates caused by invalid changes.

9.1.1.1.3 Removed features

- Remove the support for Backup and Restore Based on EBS Volume Snapshots.
- Remove the `tidb-scheduler` component.
- Remove the following CRDs: `TiDBInitializer`, `TiDBDashboard`, `DMCluster`, `FedVolumeBackup`, `FedVolumeBackupSchedule`, and `FedVolumeRestore`.
- Remove the `TiDBMonitor` and `TiDBNGMonitoring` CRDs. Related features are integrated through other methods. For details, see Deploy Monitoring and Alerts for a TiDB Cluster.

9.1.1.2 Acknowledgments

Thanks to all the developers and community members who contributed to TiDB Operator! We look forward to your feedback and suggestions to help us improve this milestone release.
