RadonDB

RadonDB MySQL Kubernetes

V2.1.4

User Guide

QingCloud Technologies

QingCloud Technologies Corp.

Issue: 01
Date: 2022-6-14

Trademark statement

 and other QingCloud trademarks are trademarks of QingCloud Technologies Corp.
All other trademarks and registered trademarks mentioned in this document remain the property of their respective owners.

Notice

Part of the products, services, and features described in this document may not be within the purchase scope or usage scope. The purchased products, services, and features shall be subjected to the commercial contracts and terms of QingCloud Technologies Corp.
This document is only for reference and will be refreshed regularly. All statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

QingCloud Technologies Corp.
Website: https://www.qingcloud.com

Preface

This document describes how to deploy and use RadonDB MySQL Kubernetes, serving as a reference for RadonDB MySQL users.

Product version

This document applies to RadonDB MySQL v2.1.4.

Intended audience

This document is intended for:

- RadonDB MySQL individual users
- RadonDB MySQL enterprise users
- O&M engineers

Change history

Issue	Date	Description
01	2022-6-8	This issue is the first official release.

Contents

1 Overview

RadonDB MySQL is an open-source, high-availability, and cloud-native cluster solution based on MySQL. It adopts the architecture of a primary database and multiple secondary databases, with a full set of management functions for security, automatic backup, monitoring and alerting, automatic storage expansion, and so on. It has been used on a large scale in the production environment by users such as banks, insurance companies, traditional large enterprises, and so on. RadonDB MySQL achieves high availability by using open-source high-availability components provided by Xenon for MySQL clusters.

While Kubernetes community users demand high-availability MySQL on Kubernetes, the RadonDB community decided to transplant RadonDB MySQL to Kubernetes and made it an open-source project in 2021. The project aims to provide an enterprise-level high-availability solution to MySQL on Kubernetes for both Kubernetes and MySQL developers.

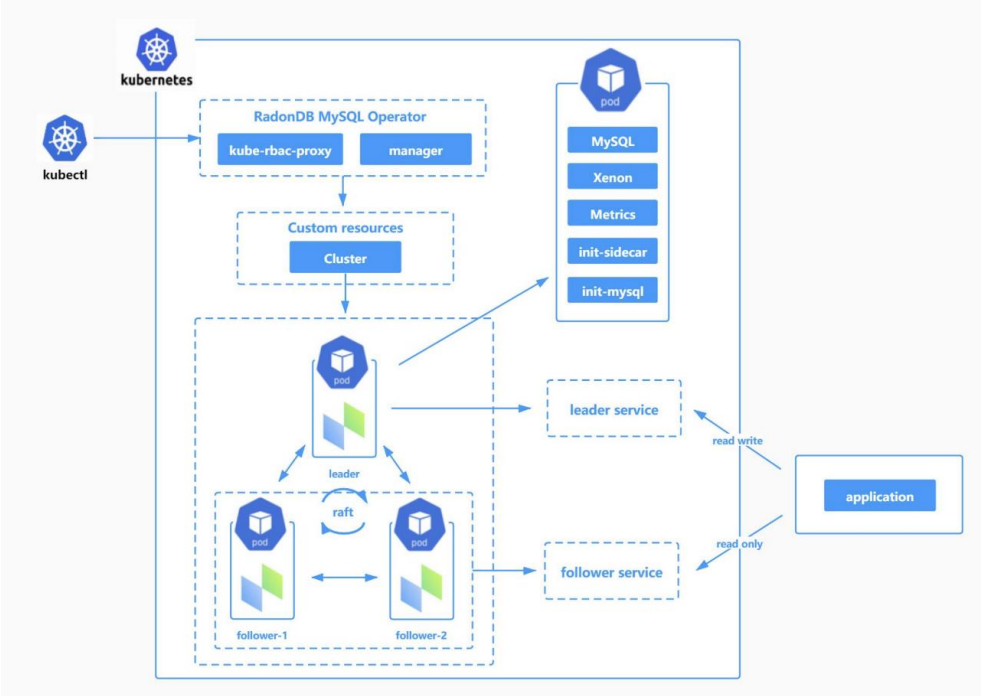
RadonDB MySQL Kubernetes supports installation, deployment, and management on Kubernetes, KubeSphere, Rancher, and other platforms, and automatically performs tasks involved in running RadonDB MySQL clusters.

1.1 Key features

- MySQL high availability
  - Automatic decentralized leader selection
  - Failover within seconds
  - Strong data consistency in cluster switching
- Cluster management
- Monitoring and alerting
- Backup
- Cluster log management
- Account management

1.2 Architecture

- Automatic decentralized leader election by Raft protocol
- Data synchronization by Semi-Sync replication based on GTID mode
- High availability with Xenon



1.3 Roadmap

1.0 Helm Chart	2.0 Operator	3.0 Operator
● High-availability MySQL ● Automatic decentralized leader election ● Failover within seconds ● Strong data consistency ● Cluster management ● Monitoring and alerting ● Cluster log management ● Account management	● Node creation/deletion ● Automatic storage expansion ● Cluster Upgrade ● Backup and recovery ● Automatic failover ● Automatic node rebuilding ● Automatic service restarting ● Account management (with APIs) ● Online migration	● Automatic O&M ● Multiple node roles ● Disaster recovery cluster ● SSL encryption

1.4 License

RadonDB MySQL is based on Apache 2.0 protocol. See License.

1.5 Strengths

- **Strong data consistency**

It adopts the high-availability architecture of one master node and multiple secondary nodes and enables automatic split-brain protection.

- **High availability**

  It meets different availability requirements with the architecture of one master node and multiple secondary nodes.

- **Automatic O&M**

  You can set strategies for automatic backup, monitoring and alerting, and automatic scaling.

- **Elastic scaling**

  CPU, memory, and storage capacity of the database are expanded according to business needs.

1.6 Use cases

- **Financial scenario**

  Strong data consistency meets the reliability requirements in financial scenarios.

- **Website O&M**

  A full set of backup, recovery, monitoring, and other O&M schemes are provided for website service requirements.

2 Deployment guide

This chapter introduces how to deploy, verify, access, and uninstall RadonDB MySQL Operator and clusters on Kubernetes.

2.1 Deployment on Kubernetes

This section describes how to deploy, verify, access, and uninstall RadonDB MySQL Operator and high-availability MySQL clusters.

2.1.1 Prerequisites

- Kubernetes cluster
- MySQL client tools

2.1.2 Procedure

Step 1 Add a Helm Repository.

Add a Helm Repository named radondb

```
$ helm repo add radondb https://radondb.github.io/radondb-mysql- kubernetes/
```

Ensure the chart named **radondb/mysql-operator** exists in the repository.

```
$ helm search repo
NAME CHART VERSION APP VERSION DESCRIPTION
radondb/mysql-operator 0.1.0 v2.1.x Open Source，High Availability Cluster，based on MySQL
```

Step 2 Deploy operator.

Set the release name to **demo** and create a Deployment named **demo-mysql-operator**.

```
$ helm install demo radondb/mysql-operator
```

> **Note**
>
> This step also creates the CRD required by the cluster.

Step 3 Deploy a RadonDB MySQL cluster.

Create an instance of the **mysqlclusters.mysql.radondb.com** CRD and thereby create a RadonDB MySQL cluster with default parameters as follows. To set cluster parameters, see Configuration Parameters.

```
$ kubectl apply -f https://github.com/radondb/radondb-mysql-kubernetes/releases/latest/download/mysql_v1alpha1_mysqlcluster.yaml
```

2.1.3 Verification

2.1.3.1 Verifying RadonDB MySQL Operator

Check the demo Deployment and its monitoring service. The deployment is successful if the following information is displayed.

```
$ kubectl get deployment,svc
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
demo-mysql-operator           1/1     1            1           7h50m

NAME                                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/mysql-operator-metrics        ClusterIP   10.96.142.22    <none>        8443/TCP   8h
```

2.1.3.2 Verifying RadonDB MySQL cluster

Run the following command to check the CRDs.

```
$ kubectl get crd | grep mysql.radondb.com
backups.mysql.radondb.com                             2021-11-02T07:00:01Z
mysqlclusters.mysql.radondb.com                       2021-11-02T07:00:01Z
mysqlusers.mysql.radondb.com                          2021-11-02T07:00:01Z
```

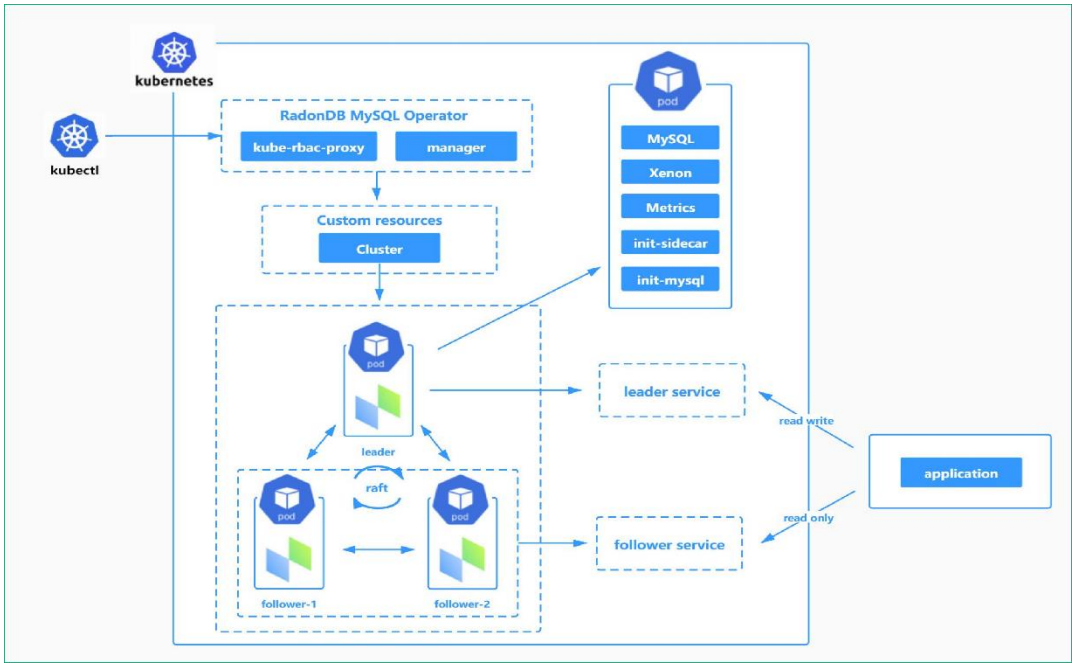
Run the following command to check the cluster. If a statefulset of three replicas (RadonDB MySQL nodes) and services used to access the nodes are displayed, the deployment is successful.

```
$ kubectl get statefulset,svc
NAME                READY   AGE
sample-mysql        3/3     7h33m

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/sample-follower       ClusterIP   10.96.131.84    <none>        3306/TCP   7h37m
service/sample-leader         ClusterIP   10.96.111.214   <none>        3306/TCP   7h37m
service/sample-mysql          ClusterIP   None            <none>        3306/TCP   7h37m
```

2.1.3.3 Accessing RadonDB MySQL

You can use **service_name** or **clusterIP** to access RadonDB MySQL in the Kubernetes cluster.

- RadonDB MySQL provides leader and follower services to access the leader node and follower nodes respectively. The leader service always points to the leader node supporting reading and writing data. The follower service always points to the read-only follower nodes.



If the client and database are in the same Kubernetes cluster, access RadonDB MySQL as follows.

Note

If the client is installed in a different Kubernetes cluster, see Access Applications in a Cluster to configure port forwarding and load balancing.

Using ClusterIP

The HA read/write IP address of RadonDB MySQL points to the **clusterIP** of the leader service, and the HA read-only IP address points to the **clusterIP** of the follower services.

```
$ mysql -h <clusterIP> -P <mysql_Port> -u <user_name> -p
```

For example, run the following command to access a leader service. The username is **radondb_usr**, and the clusterIP of the leader service is **10.10.128.136**.

```
$ mysql -h 10.10.128.136 -P 3306 -u radondb_usr -p
```

Using service_name

Pods in the Kubernetes cluster can access RadonDB MySQL by using **service_name**.

Note

service_name cannot be used to access database pods from the host machines in the Kubernetes cluster.

- Access the leader service (RadonDB MySQL leader node).

```
$ mysql -h <leader_service_name>.<namespace> -u <user_name> -p
```

For example, run the following command to access the leader service. The username is **radondb_usr**, the release name is **sample**, and the namespace of RadonDB MySQL is **default**.

```
$ mysql -h <follower_service_name>.<namespace> -u <user_name> -p
```

For example, run the following command to access the follower service. The username is **radondb_usr**, the release name is **sample**, and the namespace of RadonDB MySQL is **default**.

```
$ mysql -h sample-follower.default -u radondb_usr -p
```

2.1.4 Uninstallation

2.1.4.1 Uninstalling Operator

Uninstall RadonDB MySQL Operator with the release name **demo** in the current namespace.

```
$ helm delete demo
```

2.1.4.2 Uninstalling RadonDB MySQL cluster

Uninstall the RadonDB MySQL cluster with the release name **sample**.

```
$ kubectl delete mysqlclusters.mysql.radondb.com sample
```

2.1.4.3 Uninstalling Custom Resources

```
$ kubectl delete customresourcedefinitions.apiextensions.k8s.io mysqlclusters.mysql.radondb.com
$ kubectl delete customresourcedefinitions.apiextensions.k8s.io mysqlusers.mysql.radondb.com
$ kubectl delete customresourcedefinitions.apiextensions.k8s.io backups.mysql.radondb.com
```

2.2 Deployment on Kubesphere

This section displays how to deploy RadonDB MySQL operator and high-availability MySQL cluster on Kubesphere.

2.2.1 Prerequisites

- You need to enable the KubeSphere App Store (OpenPitrix).
- You need to create a workspace, project, and user. For more information, see Create Workspaces, Projects, Users, and Roles.
    - During installation, log in to the Web console as **admin** and operate in the **demo-project** of the demo workspace.
- You need to enable a gateway for external access.

2.2.2 Procedure

Step 1 Add an app repository.

1. Log in to the KubeSphere Web console.
2. In the **demo** workspace, go to **App Repositories** under **App Management**, and then click **Add**.
3. In the dialog displayed, specify an app repository name and add your repository URL.
   - Specify **radondb-mysql-operator** as the app repository name.
   - Add **https://radondb.github.io/radondb-mysql-kubernetes/** as the repository URL. Click **Validate** to verify the URL.

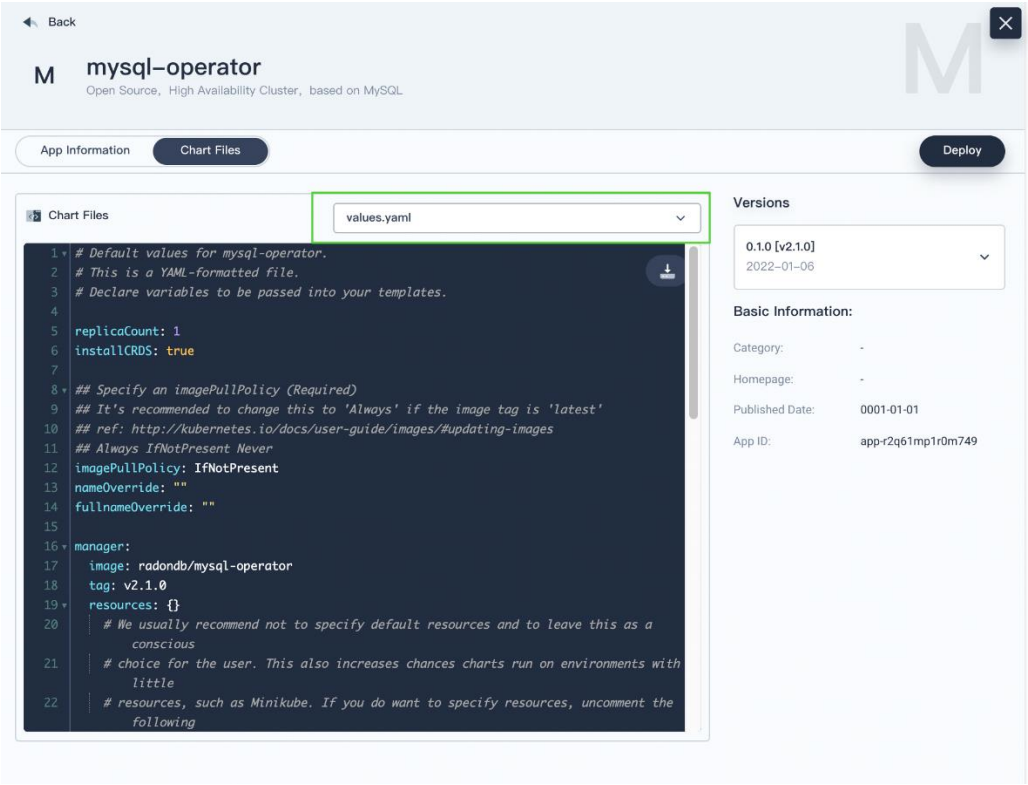
- A green check mark is displayed next to the URL if it is available. Click **OK** to continue.

4. Your repository is displayed in the repository list after being imported.

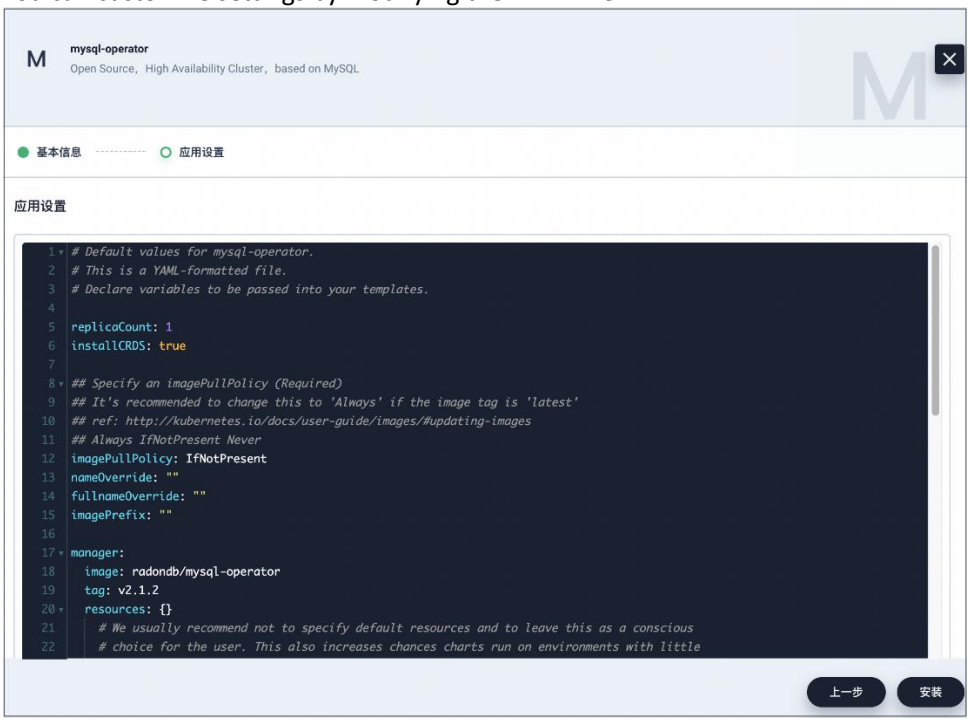
Step 2 Deploy RadonDB MySQL Operator.

1. In **demo-project**, go to **Apps** under **Application Workloads** and click **Deploy New App**.
2. In the dialog displayed, select **From App Template**.
3. On the new page, select **radondb-mysql-operator** from the drop-down list.
4. Click the **icon** of **mysql-operator**, check, and config RadonDB MySQL Operator.

5. On the **Chart Files** tab, you can view the configuration and edit the **YAML** files.
   On the **Version** list, you can view the app versions and select a version.



6. Click **Deploy**, and go to the **Basic Information** page.
   Confirm the app name, app version, and deployment location.
7. Click **Next** to continue, and go to the **App Configuration** page.
   You can customize settings by modifying the YAML file.



8. Click **Deploy**, and return to the **App Template** page. The application is successfully deployed when the application status changes to **running**.

Update Operator.

If a historical version of Operator has been deployed on Kubesphere, you can update it to the latest version as follows.

1. Delete the historical version on the Kubesphere platform.

2. Install the latest Operator with previous steps.
3. Run the following command to update the CRD. Take updating CRD to version 2.1.2 as an example:

```
kubectl apply -f https://raw.githubusercontent.com/radondb/radondb-mysql-kubernetes/v2.1.2/charts/mysql-operator/crds/mysql.radondb.com_mysqlclusters.yaml
```

Step 3 Deploy a RadonDB MySQL cluster.

You can deploy a cluster by referring to the RadonDB MySQL sample, or customizing the YAML file.

Take **mysql_v1alpha1_mysqlcluster.yaml** template as an example to create a RadonDB MySQL cluster.

1. Hover your cursor over the hammer icon in the lower right corner, and then select **Kubectl**.
2. Run the following command to install the RadonDB MySQL cluster.

```
kubectl apply -f https://github.com/radondb/radondb-mysql-kubernetes/releases/latest/download/mysql_v1alpha1_mysqlcluster.yaml --namespace=<project_name>
```

Note

When no project is specified, the cluster will be installed in the **kubesphere-controls-system** project by default. To specify a project, the install command needs to add the **--namespace=<project_name>** field.

You can see the expected output below if the installation is successful.

```
$ kubectl apply -f https://github.com/radondb/radondb-mysql-kubernetes/releases/latest/download/mysql_v1alpha1_mysqlcluster.yaml --namespace=demo-project
mysqlcluster.mysql.radondb.com/sample created
```

3. You can run the following command to view all services of the RadonDB MySQL cluster.

```
kubectl get statefulset,svc
```

Expected output

```
$ kubectl get statefulset,svc
NAME                                 READY   AGE
statefulset.apps/sample-mysql        3/3     10m

NAME                           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/default-http-backend   ClusterIP   10.96.69.202    <none>        80/TCP     3h2m
service/sample-follower        ClusterIP   10.96.9.162     <none>        3306/TCP   10m
service/sample-leader          ClusterIP   10.96.255.188   <none>        3306/TCP   10m
service/sample-mysql           ClusterIP   None            <none>        3306/TCP   10m
```

Step 4 View the status of the RadonDB MySQL cluster.

1. In **demo-project**, go to **Services** under **Application Workloads** for the information of

services.

2. Go to the **Workloads** page under **Application Workloads** and click the **StatefulSets** tab for the cluster status.
   Click a StatefulSet to go to its detail page and click the **Monitoring** tab to see the metrics in line charts over a period.

3. Go to the **Pods** page under **Application Workloads** for the node status.

4. Go to the **Volumes** page under **Storage** to check volume usage. Clicking a data node to view the monitoring information, including the current storage capacity and remaining storage.

Step 5 Configure parameters.

Container

Parameter	Description	Default
MysqlVersion	MySQL version	5.7
MysqlOpts.RootPassword	MySQL root user password	""
MysqlOpts.User	Default MySQL username	radondb_usr
MysqlOpts.Password	Default MySQL user password	RadonDB@123
MysqlOpts.Database	Default database name	radondb
MysqlOpts.InitTokuDB	TokuDB enabled	true
MysqlOpts.MysqlConf	MySQL Configuration	-
MysqlOpts.Resources	MySQL container resources	Reserve: CPU 100M, memory 256Mi; limit: CPU 500M, memory 1Gi
XenonOpts.Image	Xenon (HA MySQL) image	radondb/xenon:1.1.5-alpha
XenonOpts.AdmitDefeatHearbeatCount	Maximum heartbeat failures allowed	5
XenonOpts.ElectionTimeout	Election timeout period (milliseconds)	10000 ms
XenonOpts.Resources	Xenon container resources	Reserve: CPU 50M, memory 128Mi; limit: CPU 100M, memory 256Mi

MetricsOpts.Enabled	Metrics (monitor) container enabled	false
MetricsOpts.Image	Metrics container image	prom/mysqld-exporter: v0.12.1
MetricsOpts.Resources	Metrics container resources	Reserve: CPU 10M, memory 32Mi; limit: CPU 100M, memory 128Mi

Pod

Parameter	Description	Default
Replicas	The number of cluster nodes. The value 0, 2, 3 and 5 are allowed.	3
PodPolicy.ImagePullPolicy	The image pull policy is only allowed to be Always / IfnNotPresent / Never.	IfNotPresent
PodPolicy.Labels	Pod labels	-
PodPolicy.Annotations	Pod annotations	-
PodPolicy.Affinity	Pod affinity	-
PodPolicy.PriorityClassName	Pod priority class name	-
PodPolicy.Tolerations	Pod toleration list	-
PodPolicy.SchedulerName	Pod scheduler name	-
PodPolicy.ExtraResources	Node resources (containers except MySQL and Xenon)	Reserve: CPU 10M, memory 32Mi
PodPolicy.SidecarImage	Sidecar image	radondb/mysql-sidecar :latest
PodPolicy.BusyboxImage	Busybox image	busybox:1.32
PodPolicy.SlowLogTail	SlowLogTail enabled	false
PodPolicy.AuditLogTail	AuditLogTail enabled	false

Persistence

Parameter	Description	Default
Persistence.Enabled	Persistence enabled	true
Persistence.AccessModes	Access mode	ReadWriteOnce
Persistence.StorageClass	Storage type	-
Persistence.Size	Size	10Gi

Example

```
apiVersion: mysql.radondb.com/v1alpha1
kind: MysqlCluster
metadata:
name: sample
spec:
replicas: 3
mysqlVersion: "5.7"
# The backupSecretName specifies the secret file name which stores S3
# Information.
# If you need S3 backup or recovery, please create backup_secret.yaml,
# uncomment the following line and specify the secret name:
# backupSecretName:
# If you want to create a MySQL cluster from S3, uncomment the following line an
#d specify the directory in S3 bucket:
# restoreFrom:
mysqlOpts:
rootPassword: "RadonDB@123"
rootHost: localhost
user: radondb_usrpassword: RadonDB@123
database: radondb
initTokuDB: true
# A simple map between strings.
# Such as:
# mysqlConf:
# expire_logs_days: "7"
mysqlConf: {}
resources:
requests:
cpu: 100m
memory: 256Mi
limits:
cpu: 500m
memory: 1Gi
xenonOpts:
image: radondb/xenon:1.1.5-alpha
admitDefeatHearbeatCount: 5
electionTimeout: 10000
resources:
requests:
cpu: 50mmemory: 128Mi
limits:
cpu: 100m
memory: 256Mi
metricsOpts:
enabled: false
image: prom/mysqld-exporter:v0.12.1
resources:
requests:
cpu: 10m
memory: 32Mi
limits:
```



```
cpu: 100m
memory: 128Mi
podPolicy:
imagePullPolicy: IfNotPresent
sidecarImage: radondb/mysql-sidecar:latest
busyboxImage: busybox:1.32
slowLogTail: false
auditLogTail: false
labels: {}
annotations: {}affinity: {}
priorityClassName: ""
tolerations: []
schedulerName: ""
# extraResources defines resources for containers except MySQL and Xenon.
extraResources:
requests:
cpu: 10m
memory: 32Mi
persistence:
enabled: true
accessModes:
- ReadWriteOnce
#storageClass: ""
size: 20Gi
```

3 Features

3.1 Monitoring and alerting

The feature is supported in RadonDB MySQL Kubernetes 2.1.0 and later versions.

3.1.1 Background

The text-based format for exposing metrics required by Prometheus has been a standard in cloud-native monitoring.
The RadonDB MySQL monitoring engine is based on Prometheus MySQLd Exporter. It scrapes RadonDB MySQL metrics with **mysqld-exporter** and visualizes the metrics by third-party platforms.
This section displays how to enable RadonDB MySQL monitoring metrics.

3.1.2 Prerequisites

- A Kubernetes or KubeSphere cluster
- RadonDB MySQL Kubernetes 2.1.0 or a later version

3.1.3 Procedure

Step 1 Configure serviceMonitor.

serviceMonitor is a parameter defining the automatic monitoring engine of RadonDB MySQL Operator. It is automatically bound to **mysqld_exporter** and Prometheus automatically after being enabled.
The **serviceMonitor** parameter contains:

```
serviceMonitor:
  enabled: true
  ## Additional labels for the serviceMonitor. It is useful when you have multiple Pr
ometheus operators running to select specific ServiceMonitors.
  # additionalLabels:
  #   prometheus: prom-internal
  interval: 10s
  scrapeTimeout: 3s
  # jobLabel:
  # targetLabels:
  # podTargetLabels:
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: mysql.radondb.com
      app.kubernetes.io/name: mysql
```


You can configure **serviceMonitor** in the **charts/mysql-operator/values.yaml** file.

Note

- When a new Operator is deployed, **serviceMonitor.enabled** is set to **true** by default. The serviceMonitor is enabled.
- If the Operator deployed for the cluster is earlier than version 2.1.0, you need to redeploy a later version of Operator.

Step 2 Configure metricsOpts.

metricsOpts is a parameter defining the RadonDB MySQL cluster monitoring. You can enable the monitoring service by configuring the parameter in the **mysql_v1alpha1_mysqlcluster.yaml** file.

metricsOpts parameter contains:

```
metricsOpts:
      enabled:  false
      image:  prom/mysqld-exporter:v0.12.1

      resources:
        requests:
          cpu:  10m
          memory:  32Mi
        limits:
          cpu:  100m
          memory:  128Mi
```

Note

metricsOpts.enabled is set to **false** by default. You can set it to **true** manually.

- To enable cluster monitoring function, set metricsOpts.enabled to **true**.
- To define the resource quota for monitoring containers, set the **resources** parameter.

Apply the configuration as follows and the following information is displayed.

```
$ kubectl apply -f config/sample/mysql_v1alpha1_mysqlcluster.yaml
cluster.mysql.radondb.com/sample created/configured
```

3.1.4 Viewing monitoring service

Viewing on client

You can view the cluster monitoring service and information of **serviceMonitor** as follows.

```
$ kubectl get service,servicemonitor

$ kubectl describe servicemonitor <serviceName>
```

- **Expected output**

```
$ kubectl get service,servicemonitor
NAME                                TYPE           CLUSTER-IP       EXTERNAL-IP
PORT(S)      AGE
service/mysql-operator-metrics      ClusterIP      10.96.242.205    <none>
8443/TC
```

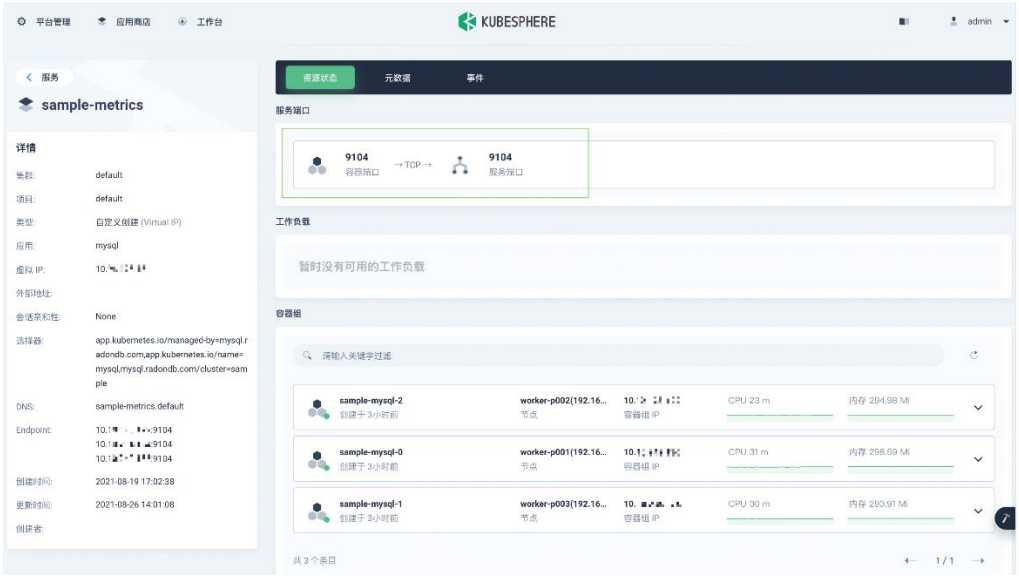


```
P      3h25m
service/sample-follower              ClusterIP      10.96.2.234       <none>            3306/T
CP     21h
service/sample-leader                ClusterIP      10.96.30.238      <none>            3306/T
CP     21h
service/sample-metrics               ClusterIP      10.96.7.222       <none>            9104/T
CP     3h24m
service/sample-mysql                 ClusterIP      None              <none>            330
6/TCP      21h

NAME                                                                                    AGE
servicemonitor.monitoring.coreos.com/demo-mysql-operator                3h25m

$ kubectl describe servicemonitor demo-mysql-operator
Name:              test-radondb-mysql-metrics
Namespace:         default
Labels:            app=test-radondb-mysql
                   app.kubernetes.io/managed-by=Helm
                   app.kubernetes.io/vendor=kubesphere
                   chart=radondb-mysql-1.0.0
                   heritage=Helm
                   release=test
Annotations:    kubesphere.io/creator: admin
API Version:    monitoring.coreos.com/v1
Kind:           ServiceMonitor
......
Spec:
  Endpoints:
    Interval:           1m
    Path:               /metrics
    Port:               metrics
    Scheme:             http
    Scrape Timeout:     10s
......
```

Viewing on KubeSphere

After the monitoring is enabled, you can view the status of the monitoring service for RadonDB MySQL Operators and clusters deployed in Kubesphere workspace.

- On the **Service** page under **Application Load** in project space, click **<clusterName>-metrics** to view the monitoring service details.



- On the **Container Group** page under **Application Load** in the project space, click a container name to view the status of metrics resources in the container.

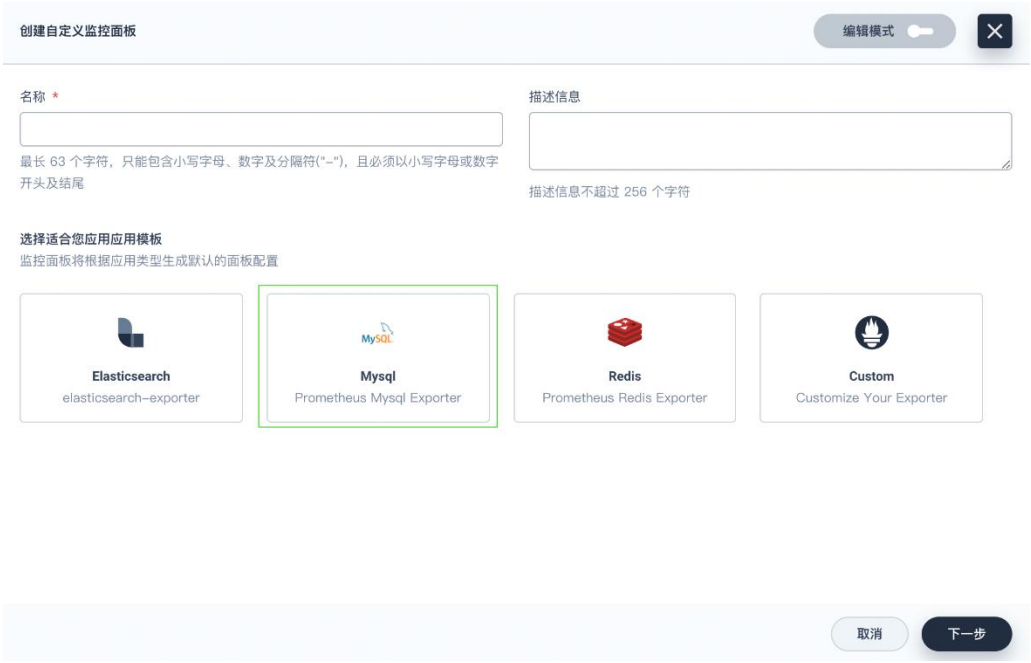
3.1.5 Viewing monitoring data

Custom application monitoring on Kubesphere

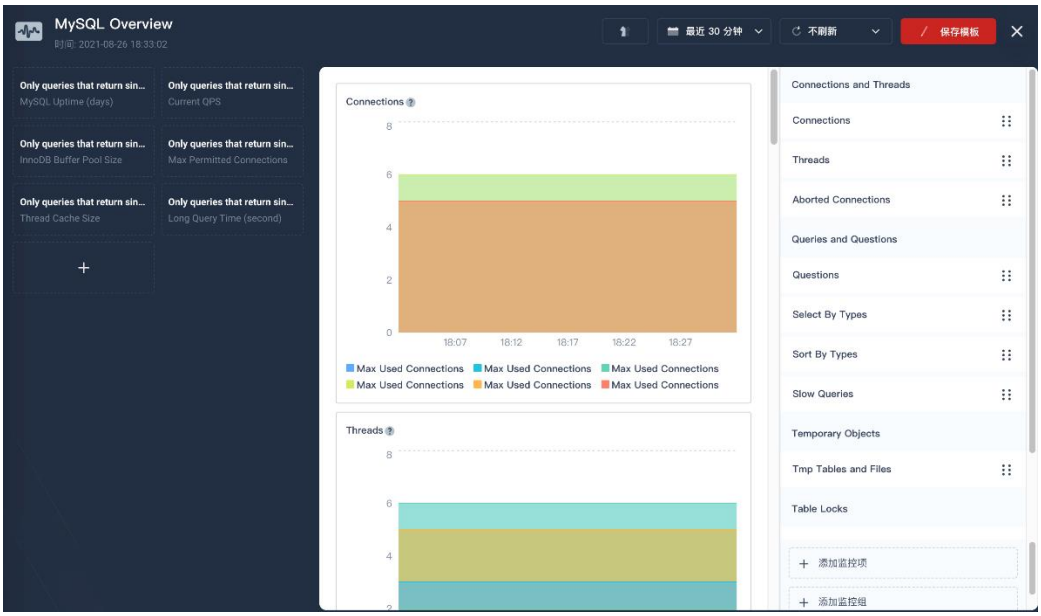
The Kubesphere monitoring engine is based on Prometheus and Prometheus Operator. Kubesphere's custom monitoring allows you to monitor and visualize RadonDB MySQL metrics.

Step 1 In the same project, go to **Custom Monitoring** under **Monitoring & Alerting** in the sidebar and click **Create**.

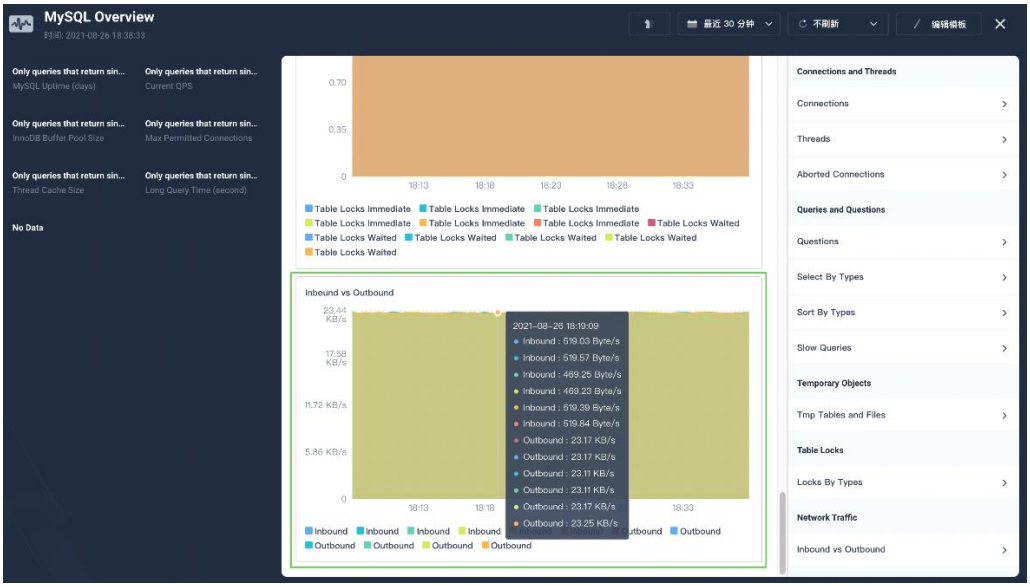
Step 2 In the displayed dialog box, set a name for the dashboard (for example, **mysql-overview**) and select the MySQL template. Click **Next** to continue.



Step 3 Click **Save Template** in the upper-right corner. A newly-created dashboard is displayed on the **Custom Monitoring Dashboards** page.



Step 4 Wait about ten minutes to view the monitoring data.

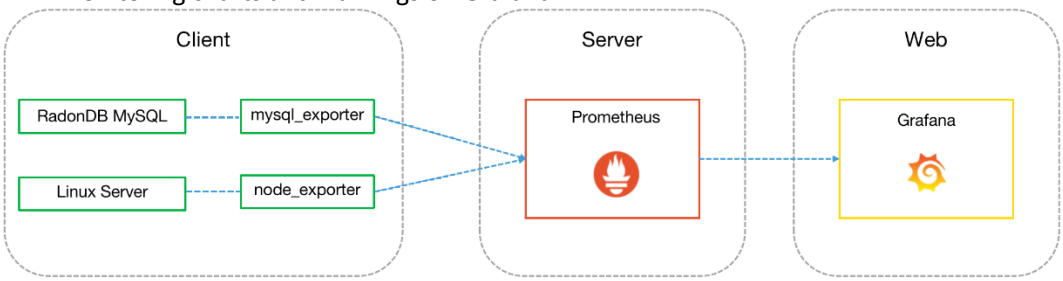


For more information, see Kubesphere custom application monitoring and visualization.

Using Prometheus and Grafana platforms

Grafana is an open-source interactive data-visualization platform. You can use Prometheus and Grafana platforms to view the monitoring information:

- Obtain the monitoring data of RadonDB MySQL services by mysql_exporter.
- Obtain the monitoring data of RadonDB MySQL servers by node_exporter.
- Transfer monitoring data to Prometheus and configure the data source to display monitoring charts and warnings on Grafana.



For more instructions on Grafana monitoring visualization, see Grafana Dashboards.

3.2 Backup and recovery

This feature is supported in RadonDB MySQL Kubernetes 2.1.0 and later versions.

3.2.1 Quick start

Step 1 Install Operator.

- Install the Operator named **test**.

```
$ helm install test charts/mysql-operator
```

Step 2 Configure the backup for S3.

- Add the secret file.

```
kind: Secret
apiVersion: v1
metadata:
  name: sample-backup-secret
  namespace: default
data:
  s3-endpoint: aHR0cDovL3MzLnNoMWEucWluZ3N0b3IuY29t
  s3-access-key: SEdKWldXVllLSENISllFRERKSUc=
  s3-secret-key: TU44TkNUdDJLdHlZREROTTc5cTNwdkxtNTlteE01blRaZlRQMWxoag==
  s3-bucket: bGFsYS1teXNxbA==
type: Opaque
```

The value **s3-xxxx** is base64-encoded. You can obtain the encoded value as follows.

```
$ echo -n "hello"|base64
```

- Create the Secret in Kubernetes.

```
$ kubectl create -f config/samples/backup_secret.yaml
```

- Add the **backupSecretName** property in **mysql_v1alpha1_mysqlcluster.yaml**.

```
spec:
  replicas: 3
  mysqlVersion: "5.7"
  backupSecretName: sample-backup-secret
  ...
```

- Create the backup file **mysql_v1alpha1_backup.yaml**.

```
apiVersion: mysql.radondb.com/v1alpha1
kind: Backup
metadata:
  name: backup-sample1
spec:
  # Add fields here
  hostname: sample-mysql-0
  clustname: sample
```

Note

hostname: The pod name in the cluster
clustername: The cluster name

Step 3 Start cluster.

```
$ kubectl apply -f config/samples/mysql_v1alpha1_mysqlcluster.yaml
```

Step 4 Start backup.

Start the backup after the cluster is successfully started.

```
$ kubectl apply -f config/samples/mysql_v1alpha1_backup.yaml
```

3.2.2 Uninstallation

3.2.2.1 Uninstalling operator

Uninstall the cluster named **test**:

```
$helm uninstall test
$kubectl delete -f config/samples/mysql_v1alpha1_backup.yaml
```

3.2.2.2 Uninstalling cluster

Uninstall the cluster named **sample**:

```
$ kubectl delete mysqlclusters.mysql.radondb.com sample
```

3.2.2.3 Uninstalling CRD

```
$ kubectl delete customresourcedefinitions.apiextensions.k8s.io mysqlclusters.mysql.radondb.com
```

3.2.3 Restore of cluster from backup

Check the S3 bucket and set the **RestoreFrom** property in the YAML file to the backup directory, for example, **backup_2021720827**.

```
...
spec:
  replicas: 3
  mysqlVersion: "5.7"
  backupSecretName: sample-backup-secret
  restoreFrom: "backup_2021720827"
...
```

Then run the following command to restore a cluster from the backup_2021720827 copy in the S3 bucket.

```
$ kubectl apply -f config/samples/mysql_v1alpha1_mysqlcluster.yaml
```

- To recover a cluster from an NFS server, operate as follows.

3.2.3.1 Creating image

```
$ docker build -f Dockerfile.sidecar -t acekingke/sidecar:0.1 . && docker push acekingke/sidecar:0.1
$ docker build -t acekingke/controller:0.1 . && docker push acekingke/controller:0.1
```

You can replace **acekingke/sidecar:0.1** with your own label.

3.2.3.2 Deploying your own image

```
$ make manifests
$ make install
$ make deploy IMG=acekingke/controller:0.1 KUSTOMIZE=~/radondb-mysql-kubernetes/bin/kustomize
```

3.3 User management with MysqlUser CRD

This feature is supported in RadonDB MySQL Kubernetes 2.1.0 and later versions.

3.3.1 Prerequisites

- The RadonDB MySQL cluster is deployed.

3.3.2 Creating user account

Step 1 Check CRD.

Run the following command, and the CRD named **mysqlusers.mysql.radondb.com** will be displayed.

```
$ kubectl get crd | grep mysqluser
mysqlusers.mysql.radondb.com                          2021-09-21T09:15:08Z
```

Step 2 Create Secret.

RadonDB MySQL uses the Secret object in Kubernetes to save user passwords.
Run the following command to create a Secret named sample-user-password using the sample configuration.

```
$ kubectl apply -f https://raw.githubusercontent.com/radondb/radondb-mysql-kubernetes/main/config/samples/mysqluser_secret.yaml
```

Step 3 Create user.

Run the following command to create a user named **sample_user** using the sample configuration.

```
$ kubectl apply -f https://raw.githubusercontent.com/radondb/radondb-mysql-kubernetes/main/config/samples/mysql_v1alpha1_mysqluser.yaml
```

> **Note**
>
> Modifying **spec.user** (username) directly creates a new user with the username. To create multiple users, make sure that **metadata.name** (CR instance name) corresponds to **spec.user**.

3.3.3 Modifying user account

The user account is defined by the parameters in the **spec** field. Currently, the following

operations are supported:

- Modify the **hosts** parameter.
- Add the **permissions** parameter.

Authorizing IP address

You are allowed to authorize the IP address of the user account by defining the **hosts** parameter:

- **%** indicates all IP addresses are authorized.
- You can modify one or more IP addresses.

```
hosts:
  - "%"
```

User privilege

You can define the database access permission for the user account with the **permissions** field in **mysqlUser**, and add user rights by adding parameters in the **permissions** field.

```
permissions:
    - database: "*"
      tables:
        - "*"
      privileges:
        - SELECT
```

- The **database** parameter indicates the database that the user account is allowed to access. ***** indicates the user account is allowed to access all databases in the cluster.
- The **tables** parameter indicates the database tables that the user account is allowed to access. ***** indicates the user account is allowed to access all tables in the database.
- The **privileges** parameter indicates the database permissions granted for the user account. For more privilege descriptions, see privileges supported by MySQL.

3.3.4 Deleting user account

Delete the MysqlUser CR created with the sample configuration as follows.

```
$ kubectl delete mysqluser sample-user-cr
```

> **Note**
> Deleting the MysqlUser CR automatically deletes the corresponding MySQL user.

3.3.5 Sample configuration

Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: sample-user-password    # Secret name, applied to the secretSelector.secret
Name
data:
```

```
  pwdForSample: UmFkb25EQkAxMjMKIA==  # secret key, applied to secretSelector.se
cretKey. The example password is base64-encoded RadonDB@123.
  # pwdForSample2:
  # pwdForSample3:
```

MysqlUser

```
apiVersion: mysql.radondb.com/v1alpha1
kind: MysqlUser
metadata:
  name: sample-user-cr  # User CR name. It is recommended that you manage one
     user with one user CR.
spec:
  user: sample_user  # The name of the user to be created/updated
  hosts:                 # Hosts can be accessed. You can specify multiple hosts. % re
presents all hosts.
        - "%"
  permissions:
    - database: "*"  # Database name. * indicates all databases.
      tables:           # Table name. * indicates all tables
          - "*"
      privileges:     # Permission. See https://dev.mysql.com/doc/refman/5.7/en/grant.
html for more details.
          - SELECT

  userOwner:  # Specify the cluster where the user is located. It cannot be modifie
d.
    clusterName: sample
    nameSpace: default # The namespace of the RadonDB MySQL cluster

  secretSelector:  # The secret key specifying the user and storing the user passwor
d
    secretName: sample-user-password  # Password name
    secretKey: pwdForSample  # Key. The passwords of multiple users can be stored
 in a secret and distinguished by keys.
```


4 Releases

4.1 Release list

The released versions of RadonDB MySQL Kubernetes are presented in reverse chronological order.

Version	Release Time
2.1.4	2022-04-07
2.1.3	2022-03-24
2.1.2	2022-02-10
2.1.1-alpha	2021-12-02
2.1.0	2021-10-26
2.0.0	2021-08-10
1.2.0	2021-06-09
1.1.0	2021-05-07
1.0	2021-04-27

For the latest release, see https://github.com/radondb/radondb-mysql-kubernetes/releases.

Recent roadmap

1. Support more ways of database backup and recovery.
2. Support more fine-grained configuration updates.
3. Support MySQL 8.0.
4. Abstract and improve external APIs.
5. Reduce the MTTR under special scenarios for better service.
6. Improve the periodic job scheduling to support repetitive jobs more efficiently.
7. Support online migration.

- Note
  - E2E testing framework is improved to cover more scenarios. Version 1.x is deployed by the Helm package manager and is not being maintained.
  - Version 2.x is implemented by Operator and is compatible with all features of version 1.x.
  - It is strongly recommended that you use the latest 2.x versions.

4.2 Version 2.1.4

Version 2.1.4 was released on April 7, 2022, mainly optimizing availability, adding Chinese and English documentation, and fixing some problems.
Thank @andyli029, @acekingke, @runkecheng, @qianfen2021, and @Patrick-LuoYu for your contributions.

Features

1. Optimize the operator availability in downtime.
2. Enable Xenon metadata persistence.
3. Add two English deployment guides.
4. Add documentation for building images.
5. Fix the inaccurate selection of headless service labels.
6. Fix version conflict in workflow static check.

Release notes

- docs: Fix typos. (#429)

Features

- chart: Optimize operator availability. (#416)
- *: Save Xenon's metadata to persistent storage. #406 (#413)

Improvements

- docs: Add tutorial of building images. #409 (#410)
- docs: Translate **deploy_radondb-mysql_operator_on_k8s.md** and **deploy_radondb-mysql_operator_on_rancher.md** (#430)
- Bug fixes
- mysqlcluser: Headless Service may select the pods of other clusters When multiple clusters. #433 (#434)

workflow: Specify version of staticcheck. #431 (#432)

4.3 Version 2.1.3

Version 2.1.3 was released on March 24, 2022, with functional optimization and upgrade based on version 2.1.2.

Acknowledgment

Thank @andyli029, @acekingke, @runkecheng, @mgw2168, and @molliezhang for your contributions.

Feature List

Achieve the one-click publish workflow.
Support rebuilding cluster nodes by labels.
Add Pod debugging mode.

Release notes

Features
workflow: Publish release only one click. #421 (#422)
mysqlcluster: Support automatic rebuild of nodes by label. (#389)
mysqlcluster: Debug Mode for Pod #375 (#383)
Improvements
.github: Adjust release-drafter (#424)
chart: Update chart version to v2.1.3. (#419)
config: Add podAntiAffinity sample yaml. #371 (#393)
docs: Add troubleshoot.md #387 (#414)
docs: Add offline deployment document. #396 (#399)
docs: Add a description of service_name connection method #401 (#402)
Bug fixes
cmd: Change HttpServer stop channel to buffered channel. #411 (#411)
status: Skip the unavailable node and set default node status. #417 (#418)
container: Add xenoncli check in the liveness probe. (#405)
syncer: Uniform use of global variables set role labels. (#394)
hack: Change Xenon's Dockerfile image branch to master. #336 (#392)

4.4 Version 2.1.2

Version 2.1.2 was released on February 17, 2022, comprehensively upgrading node reconstruction, addition and deletion, and so on.

Acknowledgment

Thank @andyli029, @acekingke, @runkecheng, and @molliezhang for your contributions.

Features

1. Support cloning data from existing nodes for initialization.
2. Support node reconstruction.
3. Support displaying Raft status of nodes.
4. Creating and deleting nodes will no longer trigger rolling updates.
5. Support one-click configuration of image address prefix.
6. Add documentation for multi-platform deployment.
7. Support E2E testing framework.

Release notes

Features
- Clone init from follower node. #322
- Support for manual repair invalid nodes. #331
- Add E2E framework and simple testcase. #347

- Support more node role labels. #334
- Support unified setting images repository address. #378
- Add tutorials of deploy radondb mysql on rancher. #338
- Add tutorials of deploy radondb mysql on kubesphere. #152

Improvements

- Upgrade E2E frame to Ginkgo v2. #360
- Update the description about access radondb mysql. #340
- Change the default path of the rbac proxy image. #146
- Make the versions provided by helm repo and release consistent. #352
- Add .gitignore about e2e logs and function. #381

Bug fixes

- Fixed the cluster status cannot be changed after the POD exit abnormally. #366
- Fixed the container time zone is not consistent with the host time zone. #329

What's changed

Full Changelog: v2.0.0...v2.1.2

4.5 Version 2.1.1

Version 2.1.1 was released on December 2, 2021.

Acknowledgment

Thank @andyli029, @runkecheng, and @molliezhang for your contributions.

Release notes

Features

- Support cloning for initialization when adding new Pods. #250 #291
- Update replicas without restart. #282
- Support displaying the Raft status of nodes in nodes. conditions. #284 #285
- charts: Support offline deployment. #300 #301
- workflow: Manage Chart using Helm repo. #290 #294
- workflow: Automatic code check and unit tests. #277
- Makefile: Synchronize the generated files to Chart while generating CRD. #280

Improvements

- syncer: Make Nodes.Conditions only show the condition of the presence node. #283 #286
- syncer: Keep PVC when closing the cluster. #304 #308
- syncer: Optimize update POD trigger conditions. #321
- sidecar: Rewrite restore logic using golang. #292 #293
- container: Optimize the directive of Mysql liveness check. #305 #318
- Dockerfile: Provide backup of district/static:nonroot image. #287 #296
- docs: Update deployment document. #298

Bug fixes

- Fix the setting method of innodb_buffer_pool_instance. #244 #265
- Fix bug of not effective version of mysql56. #203 #217
- Fix failed to restore from backup after extending pvc. #370 #291
- syncer: Fix bug of parallel updated nodes. #310 #314
- syncer: Fix operator restart when closing cluster. #312 #315
- container: Fix pod exception restart when high pressure. #305 #318
- docs: Fix check CRD about mysqluser. #281

4.6 Version 2.1.0

Version 2.1.0 was released on October 22, 2021, the fourth release of RadonDB MySQL Kubernetes, and the second version implemented by the operator.

Acknowledgment

Thank @hustjieke, @zhyass, @runkecheng, @acekingke, and @molliezhang for your contributions.

Features

1. Add monitoring function for MySQL cluster service.
After the monitoring function is enabled, a monitoring service is created and automatically connected to Prometheus.
2. Backup and restore database based on S3.
With the bucket and API key stored in S3 object storage, you can directly back up the Pod database to S3 object storage or restore a new database cluster from the backup in S3 object storage.
3. Optimize account management.
Manage MySQL users by CR. Creating, deleting and modifying CR automatically changes corresponding users. Access to databases and tables can be granted.
4. Support dynamic disk expansion.
The YAML storage capacity can be changed and automatically expanded, and the database cluster can be automatically updated.
5. Optimize start-stop logic.
6. Enrich cluster status.
Support displaying intermediate cluster status, for example: initializing, updating; new cluster status closed.
7. Support access from external services.
8. Optimize code and provide updates.
9. Improve unit testing.
10. Automatic image building, format checking and unit testing are supported by the rich workflow and Travis CI.

Release notes

Features
- Clone init from follower node. #322
- Support for manual repair invalid nodes. #331
- Add E2E framework and simple testcase. #347
- Support more node role labels. #334
- Support unified setting images repository address. #378
- Add tutorials of deploy radondb mysql on rancher. #338
- Add tutorials of deploy radondb mysql on kubesphere. #152

Improvements
- Upgrade E2E frame to Ginkgo v2. #360
- Update the description about access radondb mysql. #340
- Change the default path of the rbac proxy image. #146
- Make the versions provided by helm repo and release consistent. [#352]
- Add .gitignore about e2e logs and function. #381

Bug fixes

- Fixed the cluster status cannot be changed after the POD exit abnormally. #366
- Fixed the container time zone is not consistent with the host time zone . #329

What's changed

Full Changelog: v2.0.0...v2.1.2

4.7 Version 2.0.0

Version 2.0.0 was released on August 10, 2021. It is deployed by Helm chart instead of the operator.

Thank @andyli029, @zhyass, @runkecheng, @acekingke, @hustjieke, and @molliezhang for your contributions.

Release notes

Improvements

- Add post-start and pre-stop script #155
- Add PreStop for xenon container #145
- Move the charts images and change the key word #140 #142
- Support roll update #133 #121
- Unit test for container, cluster #131 #130
- Add the document about the deployment of operator version #132 #127
- Update the path of helm chart #126 #129
- Update mysql version to 5.7.34 #124 #123
- Add status api to support update the cluster status #120 #119
- Add operator sidecar #120 #117
- Update the config files, helm files, the Dockerfile, Makefile #120
- Update kubebuilder from v2 to v3 #114 #113
- Modify the repo #112
- Adjust the dir for operator #111
- Add operator init #123 #109
- Add rolling update feature code annotation #165
- Add ignore dir vendor and testbin #153 #154

Bug fixes

- Fix the auditLog container #181 #179
- Fix the incorrect description about MetricsOpts #177
- Fix the bug about PostStartHookError that command sh -c /scripts/post-start.sh exited with 126 #171
- Fix the path from docker to radondb #167
- Fix the bug about the pods's status when the yaml have been changed #166 #164 #161 #158
- Fix the bug that xenoncli cannot create user #163 #162
- Fix the bug about reflect.SliceHeader vet error when go 1.16.6 #141 #139
- Move the init.sql to mysql config dir radondb #128
- Fix the bug that innodb_buffer_pool_size cannot be set correctly when its size greater than int32 #125

4.8 Version 1.0

RadonDB MySQL kubernetes 1.0.0/1.1.0/1.2.0 were deployed by Helm chart.

> **Note**
>
> The 1.x versions are not being maintained. It is strongly recommended that you use the latest version!

Acknowledgment

Thank @andyli029, @zhyass, @runkecheng, @hustjieke, @molliezhang, and @KID-G for your contributions.

1.2.0 Release notes

Improvements

- Move dockerfile to dockerfiles #108
- Update logo_radondb.png and modify files #110
- Add wechart community pic #107
- Remove the step to configure-docs for the root password #105
- Update the architecture figure #102

Bug fixes

- Modify deploy links #99
- Fix some errors adjust some descriptions in README #96

1.1.0 Release notes

Improvements

- Add table content for each file #98
- Add deploy links on README.md and README_zh.md #97
- Split the deploy-document according to the different deployment methods #95 #94
- TEST Issue template #92
- Add pull request and issue templates #91 #90
- Add the document to deploy radondb-mysql #89 #49 #45
- Add the network configuration document of the service #85
- Support the feature for k8e app #83
- Rename xenondb to radondb-mysql #77 #75 #74
- Modify the key word #73 #47 #41
- Add the README.md and README_zh.md #63 #57 #55 #50 #48 #42 #37
- Support the feature for k8s #62
- Rename krypton to xenondb #40 #36
- Add publishNotReadyAddresses param in headless service #34
- Add CMD about Kubernetes #29 #21 #20 #17
- Add directory about test #16
- Support view mysql slow log #14
- Support 1 replica #13 #11
- Support read/write splitting #9
- Add the Steps about setup service for client to write/read #8
- Add remove lost+found in charts file #5
- Update the NOTES.txt #64 #3
- Add charts and dockerfile #34 #23 #18 #15 #1

Bug fixes

- Fix the error file name #93
- Modify the description in charts file #81 #66 #67 #68
- Modify the community info in READMME.md #78 #70 #69 #61 #60 #59 #52 #51
- Fix xenon error log #33 #32

- Fix the jump #31
- Fix the bug about sysbench FATAL: mysql_stmt_prepare() failed #25
- Fix the bug about hang when run cmd kubectl delete pv #24
- Fix the error about lint #22
- Fix the bug that execute sql with no response #18
- Fix the bug that slave-pod failed to initialize relay log info structure from the repository #12 #10
- Fix the path bug #7
- Fix the bug that install helm failed #4

1.0.0 Release notes

XenonDB is a High-availability cluster solution based on MySQL.

- Non-centralized automatic leader election.
- Second level switch
- Strongly consistent data
- Cluster management
- Logs, Monitoring, and alerting
- Account management

Note

XenonDB was the name of an earlier project and was later renamed RadonDB MySQL Kubernetes.

5 Glossary

API	Application Programming Interface
CI	Continuous Integration
CPU	Central Processing Unit
CRD	Custom Resource Definition
E2E	End-to-end
GTID	Global Transaction Identifier
HA	High Availability
O&M	Operations and Maintenance
SSL	Secure Sockets Layer
S3	Simple Storage Service