

中间件数据服务

DCE 5.0 针对实际应用场景，精选了一些经典的中间件来处理数据，能够满足各类应用场景的开发和维护。您可以按需安装/启用这些中间件，即插即用。

!!! tip

提升数据库性能，实现高可用和强大的扩展性，尽在 DCE 5.0 容器化赋能的中间件！

- : simple-elasticsearch:{ .lg .middle } **Elasticsearch 搜索服务**
-

这是目前全文搜索引擎的首选，它可以快速地存储、搜索和分析海量数据。

- [什么是 Elasticsearch](#)
- [ES 集群容量规划](#)
- [创建/更新/删除 ES 实例](#)
- [跨节点迁移 ES 数据](#)
- [ES 故障排查](#)

- : simple-minio:{ .lg .middle } **MinIO 对象存储**
-

一款热门、轻量、开源的对象存储方案，完美兼容 AWS S3 协议，友好支持 K8s。

- [什么是 MinIO](#)
- [创建/更新/删除 MinIO 实例](#)
- [查看 MinIO 日志](#)
- [MinIO 实例监控](#)
- [MinIO 身份管理](#)

- : simple-mysql:{ .lg .middle } **MySQL 数据库**
-

这是应用最广泛的关系数据库，具有高吞吐、低延迟、可扩展等特性。

- [MySQL 功能说明](#)
- [创建/更新/删除 MySQL 实例](#)
- [查看 MySQL 日志](#)
- [MySQL 故障排查](#)

• : simple-mongodb:{ .lg .middle } **MongoDB 数据库**

MongoDB 是一种面向文档的 NoSQL 数据库管理系统，它以灵活的数据模型和可扩展性而闻名。

- [什么是 MongoDB](#)
- [创建/更新/删除 MongoDB 实例](#)
- [查看 MongoDB 日志](#)

• : simple-postgresql:{ .lg .middle } **PostgreSQL 数据库**

DCE 5.0 采用容器化技术将 PostgreSQL 实例打包在一个独立的环境中运行。

- [什么是 PostgreSQL](#)
- [创建/更新/删除 PostgreSQL 实例](#)
- [查看 PostgreSQL 日志](#)
- [PostgreSQL 反亲和设置](#)

• : simple-redis:{ .lg .middle } **Redis 缓存服务**

这是一种内存数据库缓存服务，兼容了 Redis 和 Memcached 两种内存数据库引擎的优点。

- [什么是 Redis](#)

- [创建/更新/删除 Redis 实例](#)
 - [查看 Redis 日志](#)
 - [跨集群数据同步](#)
- : simple-rabbitmq:{ .lg .middle } **RabbitMQ 消息队列**
-

这是基于高级消息队列协议 (AMQP) 构建的消息代理软件 (亦称面向消息的中间件)，常用于交易数据的传输管道。

- [什么是 RabbitMQ](#)
 - [创建/更新/删除 RabbitMQ 实例](#)
 - [查看 RabbitMQ 日志](#)
 - [RabbitMQ 数据迁移](#)
 - [RabbitMQ 自定义插件](#)
- : simple-apachekafka:{ .lg .middle } **Kafka 消息队列**
-

这是支持流式数据处理等多种特性的分布式消息流处理中间件，常用于消息传输的数据管道。

- [什么是 Kafka](#)
 - [创建 Kafka 实例](#)
 - [查看/更新/删除 Kafka](#)
 - [Kafka 日志及实例监控](#)
 - [Kafka 参数配置](#)
- : simple-apacherocketmq:{ .lg .middle } **RocketMQ 消息队列**
-

这是基于高级消息队列协议 (AMQP) 构建的消息代理软件 (亦称面向消息的中间件)，上海道客网络科技有限公司

常用于交易数据的传输管道。

- [什么是 Rocketmq](#)
- [创建/更新/删除 Rocketmq 实例](#)

中间件学习路径

!!! info

上述中间件的文档结构大致相同，您可以参考以下学习路径。

```
graph TD
```

```

ws([选择工作空间]) --> create{部署/创建<br/>中间件实例}
create -.-> update[更新/删除实例]
create -.-> overview[实例概览]
create -.-> monitor[实例监控]
create -.-> migrate[数据管理<br/>迁移和同步]
create -.-> practice[最佳实践<br/>和故障排查]

classDef plain fill: #ddd,stroke:#fff,stroke-width:1px,color:#000;
classDef k8s fill: #326ce5,stroke:#fff,stroke-width:1px,color:#fff;
classDef cluster fill: #fff,stroke:#bbb,stroke-width:1px,color:#326ce5;

class update,overview,monitor,migrate,practice plain
class ws,create k8s

```

功能列表

	Kafka	Elastic	MySQL	Redis	PostgreSQL	MongoDB	RabbitMQ	RocketMQ	MinIO
		search		Redis	eSQL	oDB	Queue	Queue	
支持	3.1.0	7.16.3	5.7.31 / 8.0.29 / 8.0.31	6.2. / 5	15.1.0	4.2.24	3.9.25	v4.5.0	RELEASE .2023-10-07T15-07-38Z
版本									
单机	[?]	N/A	[?]	[?]	[?]	N/A	[?]	[?]	[?]
部署	[?]	[?]	[?]	[?]	[?]	[?]	[?]	[?]	[?]
高可用	[?]	[?]	[?]	[?]	[?]	[?]	[?]	[?]	[?]

	Kafka	Elastic search	MySQL	Redis	PostgreSQL	MongoDB	RabbitMQ	RocketMQ	MinIO
用部署									
垂直									
扩容	?	?	?	?	?	?	?	?	N/A
垂直	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
缩容									
水平									
扩容	?	?	?	?	?	?	?	?	?
水平	?	?	?	?	?	?	?	?	?
缩容									
存储									
扩容	?	?	?	?	?	?	?	?	?
重启	?	?	?	?	?	?	?	?	?
访问	?	N/A	?	?	?	N/A	N/A	N/A	N/A
白名单									
单									
监控	?	?	?	?	?	?	?	?	?
日志	?	?	?	?	?	?	?	?	?
参数	?	N/A	?	?	?	?	N/A	?	?
配置									
内置									
	N/A	?	?	?	N/A	N/A	N/A	N/A	N/A

	Kafka	Elastic search	MySQL	Redis	postgreSQL	MongoDB	RabbitMQ	RocketMQ	MinIO
参数									
模板									
备份	N/A	N/A	?	?	?	N/A	N/A	N/A	N/A
从备	N/A	N/A	?	?	N/A	N/A	N/A	N/A	N/A
份恢									
复									
控制	?	?	?	?	?	?	?	?	?
台									
节点	?	?	?	?	?	?	?	?	?
亲和									
性									
工作	?	?	?	?	N/A	?	?	?	N/A
负载									
反亲									
和									

!!! note

Redis 备份及恢复功能与部署版本有关。单机模式不支持备份/恢复；集群模式支持备份，不支持恢复；主备模式支持备份/恢复。

数据服务中间件与 Kubernetes 版本适配

下列表格目前仅更新了九个中间件的 Kubernetes 版本适配情况，后续会持续更新。表格

中“?”表示已进行测试且兼容，“N/A”表示该对应版本未进行测试。

版 本/	RabbitMQ	Elasticsearch	MysQL	Redis	Kafka	MinIO	PostgreSQL	MongoDB	RocketMQ
目									
1.30 .0	?	?	?	?	?	?	?	?	?
1.29 .4	?	N/A	?	?	?	?	?	?	?
1.29 .2	?	N/A	?	?	?	?	?	?	?
1.29 .1	?	N/A	?	?	?	?	?	?	?
1.29 .0	?	?	?	?	?	?	?	?	?
1.28 .9	?	N/A	?	?	?	?	N/A	N/A	?
1.28 .7	?	N/A	?	?	?	?	N/A	N/A	?
1.28 .6	?	N/A	?	?	?	?	N/A	N/A	?
1.28 .0	?	?	?	?	?	?	?	?	?
1.27 .3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.27 .2	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.27 .1	?	?	?	?	?	?	?	?	?
1.26 .6	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.26 .4	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.26 .3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.26 .2	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.26 .0	?	?	?	?	?	?	?	?	?
1.25 .11	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?

版 本/	Rabbit MQ	Elasticsear ch	Mys QL	Redis	Kafka	MinIO	PostgreS QL	Mongo DB	Rocket MQ
项 目									
1.25 .9	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.25 .8	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.25 .3	?	?	?	?	?	?	?	?	?
1.25 .2	?	?	?	?	?	?	?	?	?
1.25 .1	?	?	?	?	?	?	?	?	?
1.25 .0	?	?	?	?	?	?	?	?	?
1.24 .15	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.24 .13	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.24 .12	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.24 .7	?	?	?	?	?	?	?	?	?
1.24 .6	?	?	?	?	?	?	?	?	?
1.24 .5	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.24 .4	?	?	?	?	?	?	?	?	?
1.24 .3	?	?	?	?	?	?	?	?	?
1.24 .2	?	?	?	?	?	?	?	?	?
1.24 .1	?	?	?	?	?	?	?	?	?
1.24 .0	?	?	?	?	?	?	?	?	?
1.23 .17	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?

版 本/	RabbitMQ	Elasticsearch	MysQL	Redis	Kafka	MinIO	PostgreSQL	MongoDB	RocketMQ
项 目									
1.23 .13	?	?	?	?	?	?	?	?	?
1.23 .12	?	?	?	?	?	?	?	?	?
1.23 .11	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.23 .10	?	?	?	?	?	?	?	?	?
1.23 .9	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.23 .8	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.23 .7	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.23 .6	?	?	?	?	?	?	?	?	?
1.23 .5	?	?	?	?	?	?	?	?	?
1.23 .4	?	?	?	?	?	?	?	?	?
1.23 .3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.23 .2	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.23 .1	?	?	?	?	?	?	?	?	?
1.23 .0	?	?	?	?	?	?	?	?	?
1.22 .17	N/A	N/A	N/A	N/A	N/A	N/A	N/A	?	?
1.22 .15	?	?	?	?	?	?	?	?	N/A
1.22 .14	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.22 .13	?	?	?	?	?	?	?	?	?

版 本/	RabbitMQ	Elasticsearch	MysQL	Redis	Kafka	MinIO	PostgreSQL	MongoDB	RocketMQ
项 目									
1.22 .12	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.22 .11	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.22 .10	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.22 .9	?	?	?	?	?	?	?	?	?
1.22 .8	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.22 .7	?	?	?	?	?	?	?	?	?
1.22 .6	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.22 .5	?	?	?	?	?	?	?	?	?
1.22 .4	?	?	?	?	?	?	?	?	?
1.22 .3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
1.22 .2	?	?	?	?	?	?	?	?	?
1.22 .1	?	?	?	?	?	?	?	?	?
1.22 .0	?	?	?	?	?	?	?	?	?

数据服务权限设计说明

数据服务模块建立在[工作空间](#)之上，数据服务的配置管理及各个中间件模块，对应工作空间各个角色的权限映射如下：

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
配置管理	配置列表	查看列	?	?	?
		表			
		配置名	?	?	?
		称搜索			
		创建配	?	?	?
		置			
		更新配	?	?	?
		置			
		删除配	?	?	?
		置			
MySQL	MySQL 实例列	查看列	?	?	?
		表			
		表			
		实例名	?	?	?
		称搜索			
		创建实	?	?	?
		例			
		更新实	?	?	?
		例配置			
		删除实	?	?	?
		例			
	MySQL 实例详	实例概	?	?	?

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		情	览		
			实例监		
		控			
			查看实		
		例配置			
		参数			
			修改实		
		例配置			
		参数			
			查看实		
		例访问			
		密码			
			查看实		
		例备份			
		列表			
			创建实		
		例备份			
			修改实		
		例自动			
		备份任			
		务			

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
			[?]	[?]	[?]
	使用备份创建				
	新实例				
备份配置管理	备份配置列表	备份配置	[?]	[?]	[?]
	创建备份		[?]	[?]	[?]
	备份配置修改		[?]	[?]	[?]
	备份配置删除		[?]	[?]	[?]
	备份配置查看		[?]	[?]	[?]
配置参数	配置参数设置	查看配置参数	[?]	[?]	[?]
	配置参数修改		[?]	[?]	[?]
	配置参数设置				
RabbitMQ	RabbitMQ 实例	查看列表	[?]	[?]	[?]
	列表	表			
	实例名称		[?]	[?]	[?]
	名称搜索				
	创建实例		[?]	[?]	[?]
	实例				

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		更新实例配置	?	?	?
		删除实例	?	?	?
		实例示例	?	?	?
RabbitMQ 实例	实例概览		?	?	?
	详情	查看	?	?	?
		实例监控	?	?	?
		控制	?	?	?
		查看实例	?	?	?
		实例配置示例	?	?	?
		参数	?	?	?
		修改实例	?	?	?
		实例配置示例	?	?	?
		参数	?	?	?
		查看实例	?	?	?
		实例访问示例	?	?	?
		密码	?	?	?
Elasticsearch	Elasticsearch 实例	查看列表	?	?	?
	实例列表	表	?	?	?
		实例名称	?	?	?
		名称搜索	?	?	?

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	创建实例		?	?	?
	实例配置		?	?	?
	删除实例		?	?	?
Elasticsearch 实例	实例概览	例	?	?	?
	实例详情	览	?	?	?
	实例监控	控	?	?	?
	查看实例	查看实	?	?	?
	实例配置	例配置	?	?	?
	参数	参数	?	?	?
	修改实例	修改实	?	?	?
	实例配置	例配置	?	?	?
	参数	参数	?	?	?
	查看实例	查看实	?	?	?
	例访问	例访问	?	?	?
	密码	密码	?	?	?
Redis	Redis 实例列表	查看列表	?	?	?
		表			

数据服务	操作对象	操作	Workspace	Workspace	Workspace
			Admin	Editor	Viewer
	实例名		[?]	[?]	[?]
	称搜索				
	创建实		[?]	[?]	[?]
	例				
	更新实		[?]	[?]	[?]
	例配置				
	删除实		[?]	[?]	[?]
	例				
Redis 实例详情	实例概		[?]	[?]	[?]
	览				
	实例监		[?]	[?]	[?]
	控				
	查看实		[?]	[?]	[?]
	例配置				
	参数				
	修改实		[?]	[?]	[?]
	例配置				
	参数				
	查看实		[?]	[?]	[?]
	例访问				
	密码				

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	备份配置管理	备份配置	?	?	?
		置列表			
		创建备份	?	?	?
		备份配置			
		修改备份	?	?	?
		备份配置			
		删除备份	?	?	?
		备份配置			
	配置参数	查看配置	?	?	?
		参数设置			
		修改配置	?	?	?
		参数设置			
Kafka	Kafka 实例列表	查看列表	?	?	?
		表			
		实例名称	?	?	?
		名称搜索			
		创建实例	?	?	?
		实例			
		更新实例	?	?	?
		实例配置			
		删除实例	?	?	?

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		例			
	Kafka 实例详情	实例概览	?	?	?
		实例监	?	?	?
		控			
		查看实	?	?	?
		例配置			
		参数			
		修改实	?	?	?
		例配置			
		参数			
		查看实	?	?	?
		例访问			
		密码			
	配置参数	查看配	?	?	?
		置参数			
		修改配	?	?	?
		置参数			
MinIO	MinIO 实例列表	查看列	?	?	?
		表			
		实例名	?	?	?

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
	称搜索				
	创建实		?	?	?
	例				
	更新实		?	?	?
	例配置				
	删除实		?	?	?
	例				
MinIO 实例详情	实例概		?	?	?
	览				
	实例监		?	?	?
	控				
	查看实		?	?	?
	例配置				
	参数				
	修改实		?	?	?
	例配置				
	参数				
	查看实		?	?	?
	例访问				
	密码				
配置参数	查看配		?	?	?

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		置参数			
		修改配	[?]	[?]	[?]
		置参数			
PostgreSQL	PostgreSQL 实例	查看列	[?]	[?]	[?]
		列表	表		
			实例名	[?]	[?]
		称搜索			
		创建实	[?]	[?]	[?]
		例			
		更新实	[?]	[?]	[?]
		例配置			
		删除实	[?]	[?]	[?]
		例			
PostgreSQL	实例	实例概	[?]	[?]	[?]
		详情	览		
			实例监	[?]	[?]
		控			
		查看实	[?]	[?]	[?]
		例配置			
		参数			
		修改实	[?]	[?]	[?]

数据服务	操作对象	操作	Workspace Admin	Workspace Editor	Workspace Viewer
		例配置			
		参数			
		查看实	?	?	?
		例访问			
		密码			
配置参数		查看配	?	?	?
		置参数			
		修改配	?	?	?
		置参数			

首次进入中间件

首次使用 DCE 5.0 的 [Elasticsearch 搜索服务](#)、[Kafka 消息队列](#)、[MinIO 对象存储](#)、[MySQL 数据](#)、[RabbitMQ 消息](#)、[PostgreSQL 数据库](#)、[Redis 数据库](#)、[MongoDB 数据库](#)、[RocketMQ 消息队列](#)等中间件时，需要选择工作空间。

有关工作空间的详细介绍，可参考[工作空间与层级](#)。

下面以 MinIO 为例介绍如何选择工作空间，其他数据服务组件同理。

前提条件

1. 在 DCE 5.0 平台上创建[工作空间](#)
2. 如需直接创建中间件实例，还需要在 DCE 5.0 容器管理模块中[创建或接入](#)集群。

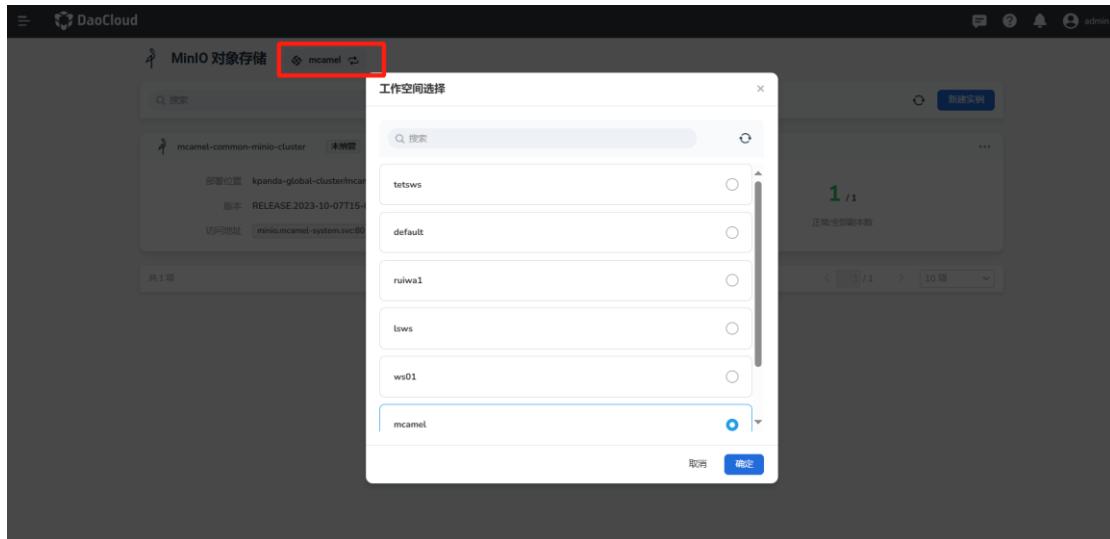
操作步骤

1. 在左侧导航栏中选择 中间件 > MinIO 存储。

MinIO 存储

MinIO 存储

2. 在弹窗中选择一个工作空间后，点击 确定。



选择工作空间

!!! note

如果未出现弹窗/想要切换工作空间，可手动点击红框中的切换图标选择新的工作空间。

3. 初次使用时，可以点击[立即部署](#)来创建 MinIO 实例。

[立即部署](#)

[立即部署](#)

数据中间件服务自定义角色权限说明

数据中间件服务模块允许用户自定义角色权限，用户可以在 [全局管理](#) 中创建[自定义角色](#)

并为角色指定各中间件的权限。

创建自定义角色时可以为角色指定三种权限：创建/编辑、查看、删除。

!!! note

部分权限选项会自动关联相关依赖权限，例如：选中 删除 权限会自动关联勾选 查看 权限，请勿取消依赖权限，避免目标权限无法正常使用。

这三种权限在数据中间件服务模块内有权执行的具体操作如下：

数据中间件服务	操作对象	操作	创建/	查	删
		编辑		看	除
配置管理	配置列表	查看列表	?	?	?
		配置名称搜索	?	?	?
		创建配置	?	?	?
		更新配置	?	?	?
		删除配置	?	?	?
MySQL	MySQL 实例列表	查看列表	?	?	?
		实例名称搜索	?	?	?
		创建实例	?	?	?
		更新实例配置	?	?	?
		删除实例	?	?	?
	MySQL 实例详情	实例概览	?	?	?
		实例监控	?	?	?
		查看实例配置参	?	?	?
		数			
		修改实例配置参	?	?	?
		数			
		查看实例访问密	?	?	?

数据中间件服务	操作对象	操作	创建/	查	删
			编辑	看	除
	码				
	查看实例备份列		?	?	?
	表				
	创建实例备份		?	?	?
	修改实例自动备		?	?	?
	份任务				
	使用备份创建新		?	?	?
	实例				
备份配置管理	备份配置列表		?	?	?
	创建备份配置		?	?	?
	修改备份配置		?	?	?
	删除备份配置		?	?	?
配置参数	查看配置参数		?	?	?
	修改配置参数		?	?	?
RabbitMQ	RabbitMQ 实例列表	查看列表	?	?	?
	实例名称搜索		?	?	?
	创建实例		?	?	?
	更新实例配置		?	?	?
	删除实例		?	?	?
	RabbitMQ 实例详情	实例概览	?	?	?

数据中间件服务	操作对象	操作	创建/	查	删
			编辑	看	除
Elasticsearch	实例监控	[?]	[?]	[?]	
Elasticsearch	查看实例配置参	[?]	[?]	[?]	
数					
Elasticsearch	修改实例配置参	[?]	[?]	[?]	
数					
Elasticsearch	查看实例访问密	[?]	[?]	[?]	
码					
Elasticsearch	查看列表	[?]	[?]	[?]	
Elasticsearch	实例名称搜索	[?]	[?]	[?]	
Elasticsearch	创建实例	[?]	[?]	[?]	
Elasticsearch	更新实例配置	[?]	[?]	[?]	
Elasticsearch	删除实例	[?]	[?]	[?]	
Elasticsearch	实例概览	[?]	[?]	[?]	
Elasticsearch	实例监控	[?]	[?]	[?]	
Elasticsearch	查看实例配置参	[?]	[?]	[?]	
数					
Elasticsearch	修改实例配置参	[?]	[?]	[?]	
数					
Elasticsearch	查看实例访问密	[?]	[?]	[?]	
码					

数据中间件服务	操作对象	操作	创建/	查	删
Redis	Redis 实例列表	查看列表	?	?	?
		实例名称搜索	?	?	?
		创建实例	?	?	?
		更新实例配置	?	?	?
		删除实例	?	?	?
	Redis 实例详情	实例概览	?	?	?
		实例监控	?	?	?
		查看实例配置参	?	?	?
		数			
		修改实例配置参	?	?	?
		数			
		查看实例访问密	?	?	?
		码			
		创建实例备份	?	?	?
		修改实例自动备	?	?	?
		份任务			
		使用备份创建新	?	?	?
		实例			
	备份配置管理	备份配置列表	?	?	?
		创建备份配置	?	?	?

数据中间件服务	操作对象	操作	创建/	查	删
			编辑	看	除
	修改备份配置	[?]	[?]	[?]	
	删除备份配置	[?]	[?]	[?]	
	查看配置参数	[?]	[?]	[?]	
	修改配置参数	[?]		[?]	
Kafka	Kafka 实例列表	查看列表	[?]	[?]	[?]
	实例名称搜索	[?]	[?]	[?]	
	创建实例	[?]	[?]	[?]	
	更新实例配置	[?]	[?]	[?]	
	删除实例	[?]	[?]	[?]	
	Kafka 实例详情	实例概览	[?]	[?]	[?]
	实例监控	[?]	[?]	[?]	
	查看实例配置参	[?]	[?]	[?]	
	数				
	修改实例配置参	[?]	[?]	[?]	
	数				
	查看实例访问密	[?]	[?]	[?]	
	码				
	配置参数	查看配置参数	[?]	[?]	[?]
	修改配置参数	[?]	[?]	[?]	
MinIO	MinIO 实例列表	查看列表	[?]	[?]	[?]

数据中间件服务	操作对象	操作	创建/	查	删
			编辑	看	除
	实例名称搜索		?	?	?
	创建实例		?	?	?
	更新实例配置		?	?	?
	删除实例		?	?	?
MinIO	实例详情	实例概览	?	?	?
		实例监控	?	?	?
		查看实例配置参	?	?	?
		数			
		修改实例配置参	?	?	?
		数			
		查看实例访问密	?	?	?
		码			
PostgreSQL	配置参数	查看配置参数	?	?	?
		修改配置参数	?	?	?
	PostgreSQL 实例列表	查看列表	?	?	?
		实例名称搜索	?	?	?
		创建实例	?	?	?
		更新实例配置	?	?	?
		删除实例	?	?	?
	PostgreSQL 实例详情	实例概览	?	?	?

数据中间件服务	操作对象	操作	创建/	查	删
			编辑	看	除
	实例监控		?	?	?
	查看实例配置参		?	?	?
	数				
	修改实例配置参		?	?	?
	数				
	查看实例访问密		?	?	?
	码				
	查看配置参数		?	?	?
	修改配置参数		?	?	?
MongoDB	MongoDB 实例列表	查看列表	?	?	?
	实例名称搜索		?	?	?
	创建实例		?	?	?
	更新实例配置		?	?	?
	删除实例		?	?	?
	MongoDB 实例详情	实例概览	?	?	?
		实例监控	?	?	?
		查看实例配置参	?	?	?
	数				
	修改实例配置参		?	?	?
	数				

数据中间件服务	操作对象	操作	创建/	查	删
			编辑	看	除
RocketMQ	查看实例访问密 码	查看实例访问密 码	?	?	?
RocketMQ 实例列表	查看列表	查看列表	?	?	?
RocketMQ 实例详情	实例名称搜索	实例名称搜索	?	?	?
RocketMQ 实例详情	创建实例	创建实例	?	?	?
RocketMQ 实例详情	更新实例配置	更新实例配置	?	?	?
RocketMQ 实例详情	删除实例	删除实例	?	?	?
RocketMQ 实例详情	实例概览	实例概览	?	?	?
RocketMQ 实例详情	实例监控	实例监控	?	?	?
RocketMQ 实例详情	查看实例配置参 数	查看实例配置参 数	?	?	?
RocketMQ 实例详情	修改实例配置参 数	修改实例配置参 数	?	?	?
RocketMQ 实例详情	查看实例访问密 码	查看实例访问密 码	?	?	?

配置管理

配置管理功能提供了对象存储的通用配置方案，用于各中间件实例的数据备份与恢复，具体配置方式如下。

1. 点击 **配置管理**，进入 **配置管理** 列表；

list
list

2. 点击 **创建** 在配置管理页面中创建一个新的配置项；

list
list

3. 在创建页面中，配置如下内容：

list
list

- **名称**：用户自定义，用于标示配置项；
- **备份类型**：该配置具有两个选项，默认为 **托管 MinIO**，将在列表中展示所有中间件 MinIO 列表中的实例；选择 **S3** 可使用外部存储，用户需要自行输入外部存储的地址，地址结构类似：<http://172.30.120.201:30456>
- **Access_Key、Secret_Key**：该项需要在 MinIO 的管理页面中获取，步骤如下：

1. 在 MinIO 实例中点击访问地址进入管理界面；

list
list

2. 点击 **Identity -> Service Account**，创建一个新的 **Service Account**；

list
list

3. 把此处创建的 **Access_Key** 和 **Secret_Key** 复制到创建配置页面。

list
list

- **Bucket 名称**：该名称用于定义备份所需的对象存储桶，可在 MinIO 管理平台中获取，如下图所示：

list
list

4. 点击 **确定** 完成创建，该配置将可用于中间件的 **备份/恢复**。

list
list

工作负载反亲和性

创建 [Elasticsearch 搜索服务](#)、[Kafka 消息队列](#)、[MinIO 对象存储](#)、[MySQL 数据](#)、[RabbitMQ 消息](#)、[PostgreSQL 数据库](#)、[Redis 数据库](#)、[MongoDB 数据库](#)、[RocketMQ 消息队列](#)实例时，可以在服务设置页面配置工作负载反亲和性。

工作负载反亲和性的作用原理是，在一定的拓扑域（作用域）范围内，如果检测到已经有工作负载带有反亲和性配置中添加的某个标签，则不会将新建的工作负载部署到该拓扑域。这样做的好处是：

- 性能优化

通过工作负载反亲和性，将实例的多个副本部署到不同的节点/可用区/区域，避免副本间的资源竞争，确保每个副本都有充足的可用资源，从而提供应用的性能和可靠性。

- 故障隔离

实例副本分布在不同的节点/可用区/区域，有效避免了单点故障问题。某个环境中的副本出现故障时，其他环境中的副本不受影响，从而保障服务整体的可用性。

操作步骤

下面以 **Redis** 为例，介绍如何设置 **工作负载反亲和性**。

!!! note

本文侧重介绍如何配置 **工作负载反亲和性**。如需了解详细的 **Redis** 实例创建过程，可参考[创建实例](../redis/user-guide/create.md)。

1. 在创建 **Redis** 实例过程中，在 **服务设置 -> 高级设置** 中启用 **工作负载反亲和性**。

创建

创建

2. 参考以下说明填写工作负载反亲和性的配置。

- 条件：分为 **尽量满足** 和 **必须满足** 两种。
 - 尽量满足：尽量尝试满足反亲和性要求，最终的部署结果不一定满足反亲和性要求
 - 必须满足：必须满足反亲和性要求。如果找不到可调度的节点/可用区/区域，则 Pod 会一直处于 Pending 状态。
- 权重：条件设置为 **必须满足** 时无需设置权重。条件设置为 **尽量满足** 时，为反亲和性规则设置权重值，优先满足权重高的规则。
- 拓扑域：拓扑域定义反亲和性的作用范围，可以为节点标签、zone 标签、region 标签，也可以由用户自定义。
- Pod 选择器：设置 Pod 标签。同一个拓扑域内，只能有一个带有此标签的 Pod。
 - 有关反亲和性以及各种操作符的详细介绍，可参考[操作符](#)

创建

创建

!!! note

上图配置的含义是，在同一个节点上只能有一个带有 `app.kubernetes.io/name` 标签且值为 `redis-test` 的 Pod。如果没有满足条件的节点，则 Pod 会一直处于 Pending 状态。

3. 参考[创建实例](#)完成后续操作。

效果验证

配置好工作负载反亲和性并且实例创建成功后，进入[容器管理](#)模块查看 **Pod** 调度信息。

[查看 Pod](#)

[查看 Pod](#)

可见一共 3 个 Pod，两个已经在正常运行并且分布在不同的节点上。

第三个 Pod 处于等待状态，点击 Pod 名称查看详情，发现原因是污点和反亲和性规则导致没有可以部署的节点。

[事件日志](#)

[事件日志](#)

!!! success

这说明前面配置的工作负载反亲和性已经生效，即一个节点上之后只能有一个带有 `app.kubernetes.io/name=redis-test` 的 Pod。如果没有满足条件的节点，则 Pod 一直处于 Pending 状态。

什么是 MySQL

MySQL 是应用最广泛的关系型数据库，是许多常见网站、应用程序和商业产品使用的主要关系数据存储，具有高吞吐、低延迟、可扩展等特性。

它是应用层协议的一个开放标准，是面向消息的中间件而设计，基于此协议的客户端与数据库来传递消息，不受产品、开发语言等条件的限制。

[创建 MySQL 实例](#)

功能特性

本页说明有关 MySQL 的功能特性。

- 高可用架构

支持主从热备架构，灵活满足各类可用性需求，稳定可靠的性能远超业界平均水准。

- 快速部署

可在线快速部署实例，图形化界面操作简便，节省采购、部署、配置等自建数据库工

作，缩短项目周期，帮助业务快速上线。

- 确保数据安全

基于实时副本技术，适配公有云、混合云和私有云，严密账号确保数据安全。

- 更低成本

可依据业务需求即时开通所需资源，无需在业务初期采购高成本硬件，有效减少初期

的资产投入，避免资源闲置浪费。

- 弹性扩缩

可依据业务压力弹性扩缩数据库资源，满足不同业务阶段对于数据库性能和存储空间

的需求。

- 自动运维

可设置自动备份策略、监控告警策略、自动扩容策略等。

在 DCE 5.0 中部署 MySQL 后，还将支持以下特性：

- 基于 Orchestrator 实现 MySQL 高可用和拓扑管理

- 支持单节点和主备模式

- 支持 phpmyadmin，提供管理页面

- 基于 mysqld-exporter 暴露指标

- 使用 Grafana Operator 集成 MySQL Dashboard，展示监控数据

- 使用 ServiceMonitor 对接 Prometheus 抓取指标

- 支持备份、恢复（依赖支持 S3 协议的存储）

MySQL 具有以下产品优势。

- 行业领先

- 高性能：MySQL 被优化以提供快速的数据检索和处理能力。它采用各种技术，如索引、缓存和查询优化，以确保高效的性能。它可以高效地处理复杂的查询和事务。

- 可靠性和稳定性：MySQL 以其稳定性和健壮性而闻名。它经过了广泛的测试，并在生产环境中有着可靠的运行记录。它提供数据完整性，并支持 ACID（原子性、一致性、隔离性、持久性）属性，以确保可靠的数据管理。

- 广泛的平台支持：MySQL 可在多个平台上使用，包括 Windows、Linux、macOS 和各种类 Unix 系统。这种灵活性使得开发人员可以根据其特定需求在不同环境中部署 MySQL。

- 社区和生态系统：MySQL 拥有庞大而活跃的开发人员和用户社区。这意味着有丰富的资源可用于支持和知识分享，如论坛、教程和用户组。此外，还有许多第三方工具和库与 MySQL 无缝集成，增强了其功能和易用性。

- 生命周期管理

- 支持图形化或通过 YAML 创建、更新和删除 MySQL 实例。
 - 支持热部署、热更新，除通用技术服务组件升级、平台升级等重大升级改造外，系统部署和升级无需停机。

- 图形化界面操作

- 支持以图形化方式创建、查看、更新、删除以及弹性扩容。管理员可以根据自己的偏好，定制图形化参数。

- 容器化消息平台

MySQL 基于 Kubernetes 和 Docker，原生支持容器化部署，将资源利用率提高到最大。

- 兼容性和开放性

- MySQL 兼容 Intel CPU 和国产化 CPU（如飞腾 ARM，华为鲲鹏 ARM 和海光 x86 等），兼容国产操作系统（如麒麟操作系统 v10，统信服务器操作系统 UOS 等），这些均有授权的电子证明和证书。
- 具有开放的体系架构，提供开放标准 API 接口保证基础能力、管理服务、日志、监控等资源和服务被高效的调度、管理与使用，支持第三方管理平台通过 API 接口等方式实现相关编排与使用。

离线升级中间件 - MySQL 模块

本页说明从[下载中心](#)下载中间件 - MySQL 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 __mcamel__ 字样是中间件模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

==== “已安装 chart repo”

若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
 url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```

```

==== “未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: LOCAL
 path: ./local-repo # chart 本地路径
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```

```

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
tar -xvf mcamel-mysql_0.11.1_amd64.tar
cd mcamel-mysql_0.11.1_amd64
tar -xvf mcamel-mysql_0.11.1.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

==== “通过 helm repo 升级”

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep mysql
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-mysql http://{harbor url}/chartrepo/{project}
```

```

1. 更新 helm 仓库。

```
```shell
helm repo update mcamel/mcamel-mysql # helm 版本过低会导致失败，若失败，请尝试执行 helm update repo
```

```

1. 选择您想安装的版本（建议安装最新版本）。

```
```shell
helm search repo mcamel/mcamel-mysql --versions
```

```none
[root@master ~]# helm search repo mcamel/mcamel-mysql --versions
NAME CHART VERSION APP VERSION DESCRIPTION
mcamel/mcamel-mysql 0.11.1 0.11.1 A Helm chart for Kubernetes
...
```

```

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-mysql -n mcamel-system -o yaml > mcamel-mysql.yaml
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 `mcamel-mysql.yaml` 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-mysql mcamel/mcamel-mysql \
-n mcamel-system \
-f ./mcamel-mysql.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.11.1
```

```
```

### ==== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-mysql -n mcamel-system -o yaml > mcamel-mysql.yaml
```

```
```

1. 执行 `helm upgrade` 。

升级前建议您覆盖 `bak.yaml` 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-mysql . \
-n mcamel-system \
-f ./mcamel-mysql.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```

```
```

# 基本概念

本页的词汇表定义了 MySQL 背后的核心概念，以帮助您构建 MySQL 工作原理的思维模型，并了解文档在使用某些术语时所指代的内容。

- ACID 合规

ACID 合规性是 Atomicity、Consistency、Isolation 和 Durability 这 4 个单词的首字母缩写，是一组由原子性、一致性、隔离性和持久性组成的数据库特性，可确保高效完成数据库事务。

- ACL

访问控制列表或 ACL 是控制对系统资源访问的用户权限列表。

- Connection Status Metric ( 连接状态指标 )

连接状态指标是衡量创建、连接和运行的线程数与数据库连接限制相关的指标。

- DBaaS ( 数据库即服务 )

数据库即服务、托管数据库服务，简称为 DBaaS，这是一种云服务，允许用户在订阅的基础上访问云数据库系统，而无需拥有个人云数据系统。

- E2EE

英文全称为 End-to-end encryption，即端到端加密，或简称为 E2EE，是一种通信系统，它为所有人加密消息和消息服务，但接收消息的用户和发送消息的用户除外。

- Failover ( 故障转移 )

故障转移是一种高可用性 (HA) 机制，可监控服务器的故障并在主服务器发生故障时将流量或操作重新路由到备用服务器。

- High Availability ( 高可用性 )

高可用性 (HA) 是一种基础架构设计方法，专注于减少停机时间和消除单点故障。

- Hot Standby ( 热备 )

热备用是侦听主节点何时发生故障以便备用节点取代其位置的行为。

- Index vs. Sequential Reads Metric ( 索引与顺序读取指标 )

索引与顺序读取指标图显示了使用索引的读取占主服务器上所有数据库（模式）的读取总数的比例。

- LUKS Disk Encryption ( LUKS 磁盘加密 )

Linux 统一密钥设置磁盘加密 (LUKS) 是 Linux 存储设备的开源磁盘加密规范。

- Machine Type ( 机器类型 )

机器类型是用于虚拟机 (VM) 实例的一组虚拟化硬件资源。

- Node Plan ( 节点计划 )

节点计划、数据库或集群配置是节点规格的硬件计划。

- Operations Throughput Metric ( 运营吞吐量指标 )

操作吞吐量指标是对服务器上所有数据库的获取、插入、更新和删除操作的吞吐量的度量。

- Point-In-Time-Recovery ( 恢复时间点 )

恢复时间点（简称 PITR）确保进行自动备份，以便恢复在服务器先前状态下创建的数据。

- Port ( 端口 )

端口是网络连接的通信端点。使用每个传输协议所用的端口号来标识一个端口。

- Read-Only Node ( 只读节点 )

只读节点是集群主节点的副本。

- SQL Mode ( SQL 模式 )

SQL 模式或 `sql_mode` 是一个 MySQL 系统变量，用于配置 MySQL 服务器的操作特性。

- SSL Certificate ( SSL 证书 )

SSL 证书是描述网站身份的数字凭证。

- Standby Node ( 备用节点 )

备用节点是在热备模式时设为空闲待用的节点。

- Tag ( 标记 )

标记是与资源相关联的关键字，有助于管理资源所有权并组织对资源的查找和操作。

## MySQL Release Notes

本页列出 MySQL 数据库的 Release Notes，便于您了解各版本的演进路径和特性变化。

\*[mcamel-mysql]: mcamel 是 DaoCloud 所有中间件的开发代号，mysql 是应用最广泛的关系型数据库

**2025-02-28**

**v0.26.0**

- 优化 升级 `mysqld-exporter` 镜像，并支持 MySQL-MGR 的告警规则

**2024-09-30**

**v0.22.0**

- 新增 支持手动切换主从节点

- **新增** MGR 实例支持配置 Router 节点
- **修复** 部分操作无审计日志的问题

## 2024-08-31

### v0.21.0

- **优化** 创建实例时不可选择异常的集群
- **优化** 接口中权限泄露的问题

## 2024-05-31

### v0.18.0

- **新增** 参数模板导入功能
- **优化** 支持批量修改实例参数
- **优化** 删除备份时可选是否删除 S3 中备份数据
- **优化** 调整 default max\_connections 参数至 512
- **修复** MySQL 工作空间接口报错
- **修复** 实例详情无法展示密码

## 2024-04-30

### v0.17.0

- **新增** MySQL 实例拓扑
- **优化** 增加命名空间配额的提示

- 优化 升级 MGR operator 到 v8.3.0-2.1.2
- 修复 安装器默认部署的 MGR 实例详情页无法展示密码和配置参数
- 修复 MGR 实例没有延时指标的问题
- 修复 实例监控看板 主从延时没有默认选中的问题

**2024-03-31**

**v0.16.0**

- 优化 当用户权限不足时无法读取 MySQL 的密码

**2024-01-31**

**v0.15.0**

- 优化 在全局管理中增加 MySQL 版本展示

**2023-12-31**

**v0.14.0**

- 新增 支持创建组复制模式的实例
- 修复 创建实例时部分输入框填写特殊字符的校验未生效的问题

**2023-11-30**

**v0.13.0**

- 新增 支持记录操作审计日志

- **优化** 实例列表未获取到列表信息时的提示
- **优化** Mcamel-MySQL 监控 Dashboard 的中英文展示

## 2023-10-31

### v0.12.0

- **新增** 离线升级
- **新增** 实例重启功能
- **新增** 工作负载反亲和配置
- **新增** 跨集群恢复实例
- **修复** Pod 列表展示地址为 Host IP
- **修复** cloudshell 权限问题

## 2023-08-31

### v0.11.0

- **新增** 参数模板功能
- **优化** KindBase 语法兼容
- **优化** operator 创建过程的页面展示

## 2023-07-31

### v0.10.3

- **新增** UI 界面的权限访问限制

## 2023-06-30

### v0.10.0

- 优化 `mcamel-mysql` 备份管理页面结构样式展示
- 优化 `mcamel-mysql` 监控图表，去除干扰元素并新增时间范围选择
- 优化 `mcamel-mysql` 存储容量指标来源，使用中立指标
- 优化 `mcamel-mysql` ServiceMonitor 闭环安装

## 2023-05-30

### v0.9.0

- 新增 `mcamel-mysql` 对接全局管理审计日志模块
- 新增 `mcamel-mysql` 可配置实例监控数据采集间隔时间
- 修复 `mcamel-mysql` 安装 MySQL Operator 多副本时，Raft 集群无法正常建立
- 修复 `mcamel-mysql` 升级 MySQL Operator 多副本时 PodDisruptionBudget 版本到 v1

## 2023-04-27

### v0.8.2

- 新增 `mcamel-mysql` 详情页面展示相关的事件
- 新增 `mcamel-mysql` openapi 列表接口支持 Cluster 与 Namespace 字段过滤
- 新增 `mcamel-mysql` 自定义角色
- 新增 `mcamel-mysql` 对接 HwameiStor，支持存储容量展示（需要手动创建 HwameiStor exporter ServiceMonitor）

- 升级 优化 调度策略增加滑动按钮

## 2023-03-28

### v0.7.0

#### 新功能

- 新增 `mcamel-mysql` 支持中间件链路追踪适配
- 新增 安装 `mcamel-mysql` 根据参数配置启用链路追踪
- 新增 `mcamel-mysql` PhpMyAdmin 支持 LoadBalancer 类型

#### 升级

- 升级 `golang.org/x/net` 到 v0.7.0
- 升级 GHippo SDK 到 v0.14.0
- 优化 `mcamel-mysql common-mysql` 支持多个实例优化
- 优化 `mcamel-mysql` 排障手册增加更多处理方法

## 2023-02-23

### v0.6.0

#### 新功能

- 新增 `mcamel-mysql helm-docs` 模板文件
- 新增 `mcamel-mysql` 应用商店中的 Operator 只能安装在 `mcamel-system`
- 新增 `mcamel-mysql` 支持 cloud shell

- **新增 mcamel-mysql 支持导航栏单独注册**
- **新增 mcamel-mysql 支持查看日志**
- **新增 mcamel-mysql 更新 Operator 版本**
- **新增 mcamel-mysql 展示 common MySQL 在实例列表中**
- **新增 mcamel-mysql 支持 MySQL8.0.29**
- **新增 mcamel-mysql 支持 LB**
- **新增 mcamel-mysql 支持 Operator 对接 chart-syncer**
- **新增 mcamel-mysql Operator 的 finalizers 权限以支持 openshift**
- **新增 UI 增加 MySQL 主从复制延迟情况展示**
- **新增 文档 增加 日志查看操作说明，支持自定义查询、导出等功能**

## 优化

- **升级 mcamel-mysql 升级离线镜像检测脚本**

## 修复

- **修复 mcamel-mysql 实例名太长导致自定义资源无法创建的问题**
- **修复 mcamel-mysql 工作空间 Editor 用户无法查看实例密码**
- **修复 mcamel-mysql 配置文件里 expire-logs-days 参数定义重复**
- **修复 mcamel-mysql 8.0 环境下 binlog 过期时间不符合预期**
- **修复 mcamel-mysql 备份集列表会展示出同名集群旧的备份集**

**2022-12-25****v0.5.0**

- **新增** NodePort 端口冲突提前检测
- **新增** 节点亲和性配置
- **新增** 创建备份配置时，可校验 Bucket
- **修复** arm 环境中无法展示默认配置
- **修复** 创建实例时 name 校验与前端不一致
- **修复** 重新接入变更名字的集群后，配置管理地址展示错误
- **修复** 保存自动备份配置失败
- **修复** 自动备份集无法展示的问题

**2022-11-08****v0.4.0****新功能**

- **新增** 增加 MySQL 生命周期管理接口功能
- **新增** 增加 MySQL 详情接口功能
- **新增** 基于 grafana crd 对接 insight
- **新增** 增加与 ghippo 服务对接接口
- **新增** 增加与 kpanda 对接接口
- **新增** 单测覆盖率提升到 30%

- **新增** 新增备份恢复功能
- **新增** 新增备份配置接口
- **新增** 实例列表接口新增备份恢复来源字段
- **新增** 获取用户列表接口
- **新增** mysql-operator chart 参数，来指定 metric exporter 镜像
- **新增** 支持 arm64 架构
- **新增** 添加 arm64 operator 镜像打包
- **新增** 支持密码脱敏
- **新增** 支持服务暴露为 nodeport
- **新增** 支持 mtls
- **新增 文档** 第一次文档网站发布
- **新增 文档** 基本概念
- **新增 文档** Concepts
- **新增 文档** 首次使用 MySQL
- **新增 文档** 删除 MySQL 实例

## 优化

- **优化** 将时间戳 api 字段统一调整为 int64
- **修复** 修复备份列表接口模糊搜索无效
- **修复** 修复依赖漏洞
- **修复** 备份 Job 被删除后，无法展示备份任务列表
- **修复** 当镜像有大写和数字时不能被抓取到的问题

**2022-10-18****v0.3**

- **新增** 增加 MySQL 生命周期管理接口功能
- **新增** 增加 MySQL 详情接口功能
- **新增** 基于 grafana crd 对接 insight
- **新增** 增加与 ghippo 服务对接接口
- **新增** 增加与 kpanda 对接接口
- **新增** 单测覆盖率提升到 30%
- **新增** 新增备份恢复功能
- **新增** 新增备份配置接口
- **新增** 实例列表接口新增备份恢复来源字段
- **修复** 修复备份列表接口模糊搜索无效
- **优化** 将时间戳 api 字段统一调整为 int64

## 创建 MySQL 实例

接入 MySQL 数据库后，参照以下步骤创建 MySQL 实例。

1. 在实例列表中，点击右上角的 **创建主备实例**。

The screenshot shows the DaoCloud MySQL Database management interface. It displays two MySQL clusters:

- mcamel-common-kpanda-mysql-cluster** (Running):
  - 部署位置: kpanda-global-cluster/mcamel-system
  - 版本: 8.0.29
  - 部署类型: 单节点
  - 副本数: 1 副本
  - 资源配置: CPU 请求数 1 Core, 内存 请求数 0.98 Gi, 磁盘 32 Gi
  - 状态: 1 / 1 正常(全部副本数)
  - 访问地址: mcamel-common-kpanda-mysql-cluster-mys...
- mcamel-common-mysql-cluster** (Running):
  - 部署位置: kpanda-global-cluster/mcamel-system
  - 版本: 5.7.31
  - 部署类型: 单节点
  - 副本数: 1 副本
  - 资源配置: CPU 请求数 1 Core, 内存 请求数 0.98 Gi, 磁盘 15 Gi
  - 状态: 1 / 1 正常(全部副本数)
  - 访问地址: mcamel-common-mysql-cluster-mys...

共 2 项 1 / 1 10 项

## 创建实例

2. 在创建主备实例页面中，配置 **基本信息** 后，点击 **下一步**。

The screenshot shows the 'Create Primary Slave Instance' wizard, Step 1: Basic Information.

**基本信息**

- MySQL 实例名称: test (必填)
- 描述: (输入框)

**部署位置**

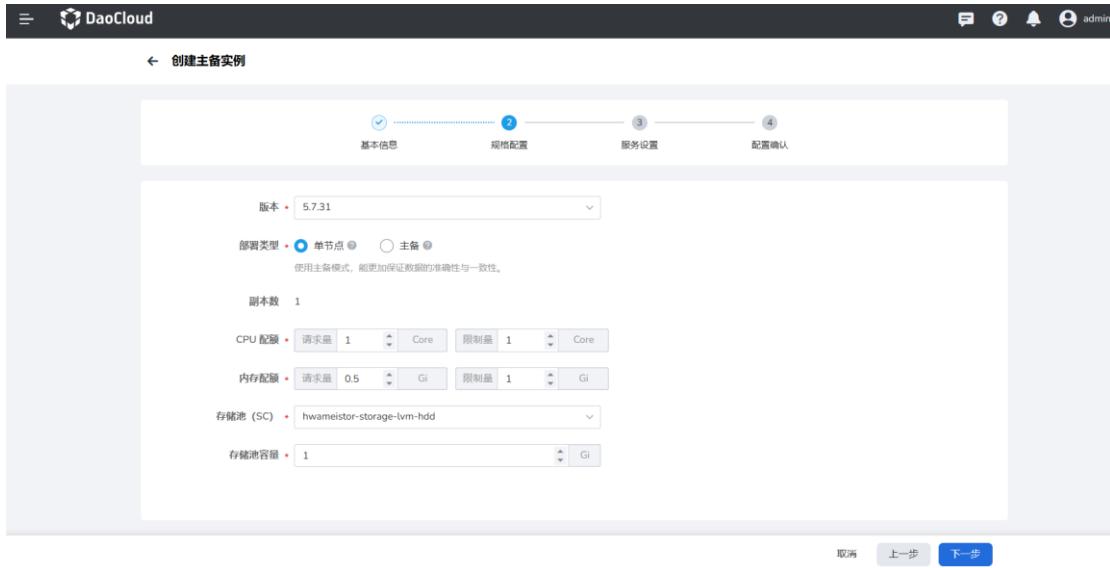
- 集群: kpanda-global-cluster (必填)
- 命名空间: chuanjia-donot-delete

安装环境检测: 集群状态检测正常，可继续安装

取消 下一步

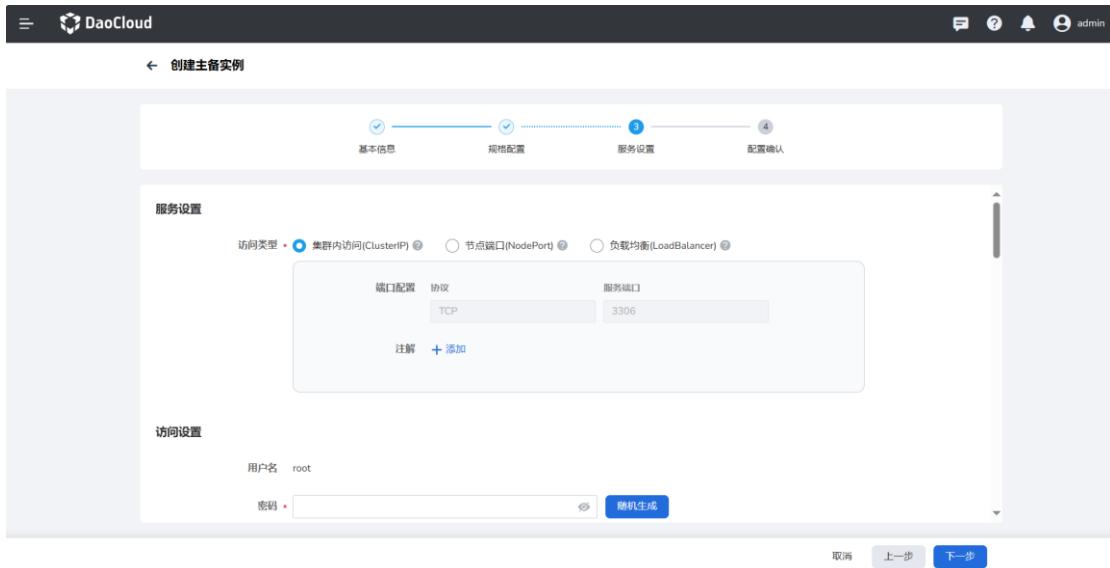
## 基本信息

3. 选择部署类型、CPU、内存和存储等 规格配置 后，点击 **下一步**。



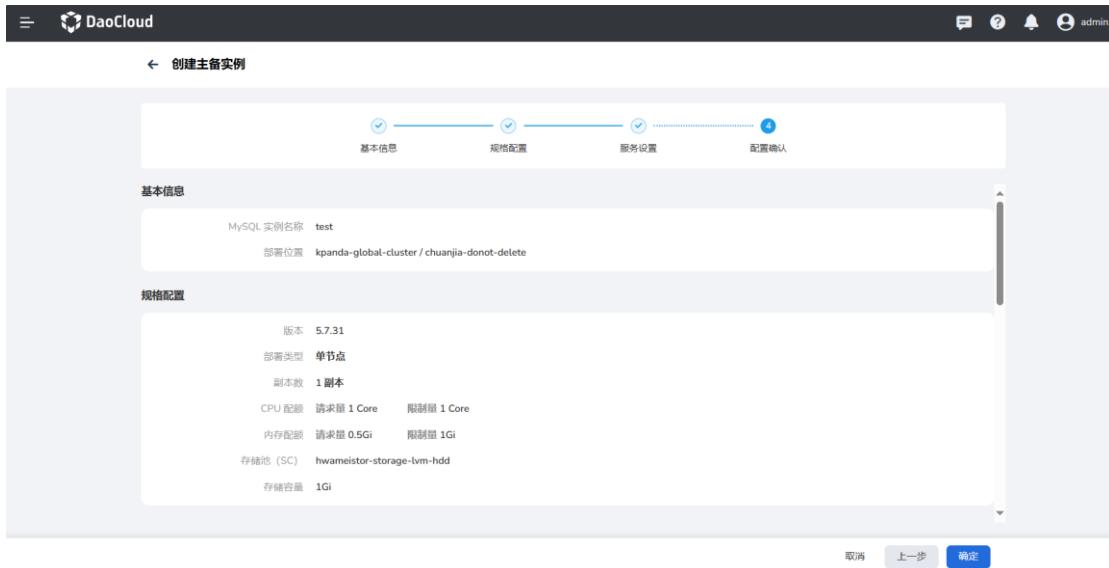
## 规格配置

4. 设置用户名和密码等 **服务设置**，默认采用 ClusterIP 作为访问方式。



## 服务设置

5. 确认基本信息、规格配置、服务设置的信息准确无误后，点击 **确定**。



确认

6. 返回实例列表，屏幕将提示 **创建实例成功**。

成功创建

成功创建

## 创建 MySQL MGR 实例

MySQL Group Replication (MGR) 是一个实现分布式数据库系统不同节点之间一致性同步的解决方案，提供了一个容错和高可用的集群环境。它使用了复制组中所有服务器的状态更新来实现高速且可靠的数据冗余。

### 前提条件

在开始设置 MGR 集群之前，需要确保以下前提条件已经满足：

- 目标集群中已提前部署 MySQL-Operator 且处于运行中状态。

## 操作步骤

1. 在实例列表中，点击下拉按钮 **创建 MGR 实例**。

2. 填写实例的基本信息，选择部署的集群和命名空间。

系统会对多选部署位置的安装前提条件进行检测，若检测不通过，请根据系统提示进行安装。

3. 选择部署实例的规格，点击 **下一步**。

- 版本：目前仅支持 8.0.31 版本

- 副本数：MGR 单主模式的副本数范围为 3 - 9，推荐使用奇数副本。

4. 在配置实例的访问设置，点击 **下一步**。

- 访问类型：选择实例对外暴露服务的访问类型

- 管理工具：开启后可以通过 phpMyAdmin 管理工具管理 MySQL

5. 确认实例的配置无误后，点击确定完成创建后返回告警模板列表。

## 更新 MySQL 实例

如果想要更新或修改 MySQL 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的  按钮，在弹出菜单中选择 **更新实例**。

**更新实例**

**更新实例**

2. 基本信息：只能修改描述。然后点击 **下一步**。

**基本信息**

**基本信息**

3. 修改规格配置后点击 **下一步**。

规格配置

规格配置

4. 修改服务设置后点击 **确定**。

服务设置

服务设置

## 删除 MySQL 实例

如果想要删除一个 MySQL 实例，可以执行如下操作：

1. 在 MySQL 实例列表中，点击右侧的  按钮，在弹出菜单中选择 **删除实例**。

删除实例

删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击 **删除** 按钮。

点击删除

点击删除

!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

## 查看实例

在 MySQL 实例列表中，选择想要查看的实例，点击实例名称进入详情页面。

## 概览

概览页面支持查看基本信息、访问设置、资源配额、主从同步延时、监控告警、容器组列表

表、最近事件等。其中，

- **主从同步延迟**：是指主从复制或同步过程中由于网络延迟或其他因素导致的从节点接收到主节点发送的数据包的时间差。
- **监控告警**：在概览页面仅支持查看最近 10 条告警，点击 **查看更多** 可跳转至告警列表。

表。

运行状态	部署位置	创建时间	版本
运行中	kpanda-global-clust...	2024-08-21 03:46	5.7.31

副本数	访问地址	用户名/密码	描述
1 副本	test-mysql-benchma...	root/******	benchmark所需, 提...

已用	总计
0.28 Core	1 Core
0.42 Gi	1 Gi

存储量
1 Gi

**概览**

## 配置参数

点击页面右上角 **更新** 即可更新 当前值。

The screenshot shows the MySQL configuration interface. On the left sidebar, under 'Backup Management', the 'Configuration Parameters' option is selected. The main panel displays a table for configuration parameters, specifically focusing on 'log-bin'. A warning message at the top states: '参数更新需要重启实例。重启会导致实例不可用，请谨慎操作!' (Parameter updates require restarting the instance. Restarting will make the instance unavailable, please operate with caution!).

## 配置参数

**!!! warning**

参数更新需要重启实例，重启会导致实例不可用，请谨慎操作！

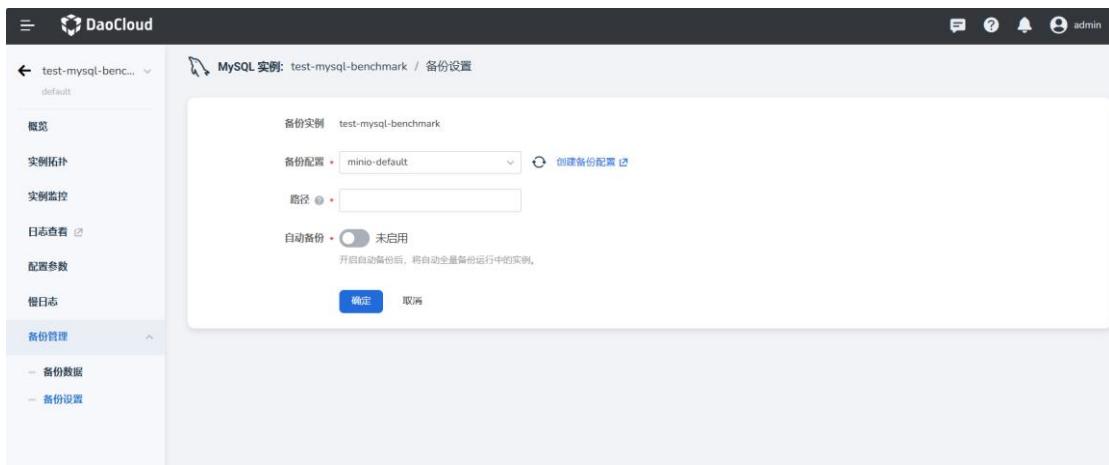
## 备份管理

- 下拉框中选择 **备份数据**，点击右上角 **创建备份**，即可创建备份数据。

The screenshot shows the MySQL backup management interface. Under 'Backup Management', the 'Backup Data' option is selected. The main panel displays a table for backup data, which currently shows '暂无数据' (No data available). A blue button labeled 'Create Backup' is visible in the top right corner.

## 备份数据

- 选择 **备份设置**，即可更改备份配置、添加路径、启用自动备份。



## 备份设置

# MySQL 手动切换主备节点

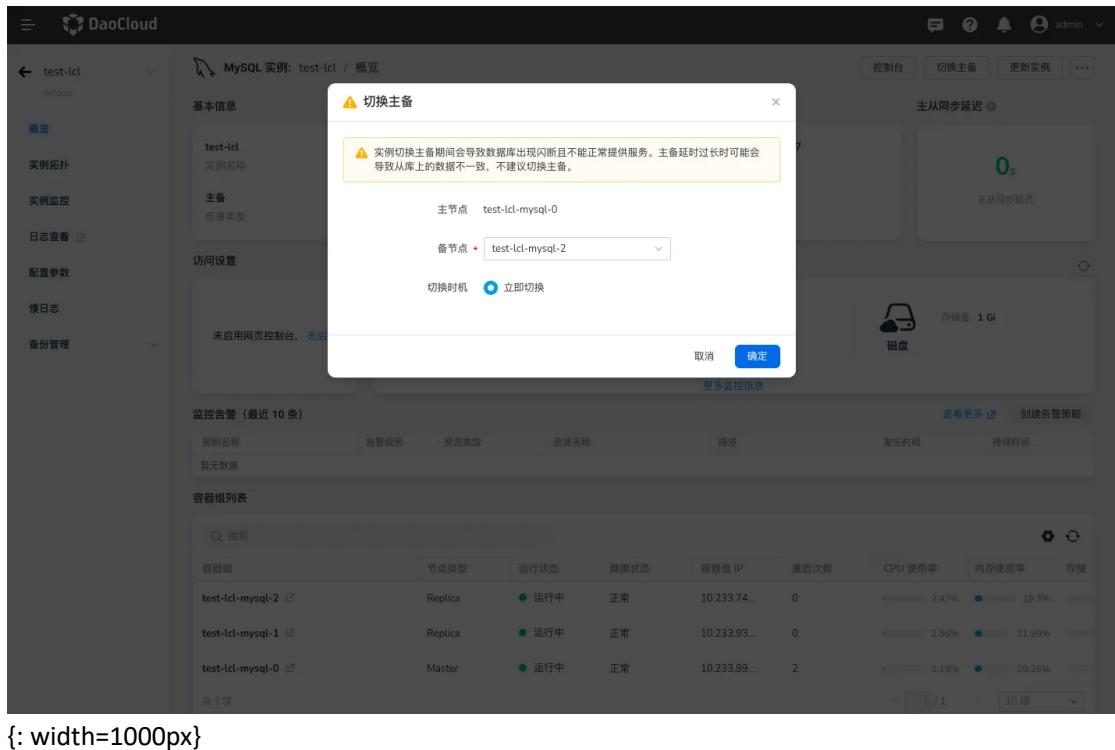
MySQL 手动切换主备功能允许数据库管理员在主服务器出现故障或需要维护时，手动将备份服务器提升为新的主服务器。通过执行适当的命令和配置，确保数据一致性和服务的连续性，从而实现高可用性。

## 注意事项

1. 确保主节点和备节点的数据一致性，避免在切换过程中出现数据丢失或不一致。
2. 在切换之前，确认实例的状态为运行中。

## 操作步骤

1. 点击进入目标 MySQL 实例的详情页
2. 在概览页的右上角，点击 **切换主备**
3. 展开备节点的下拉框，可以指定升级为主节点的备节点



## 查看 MySQL 日志

### 操作步骤

通过访问每个 MySQL 的实例详情，页面；可以支持查看 MySQL 的日志。

1. 在 MySQL 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。

image  
image

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。

image  
image

3. 点击 [日志查看](#) 即可进入到日志查看页面 ([Insight](#) 日志查看)。

## 日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围  
( 可查看的日志范围以 可观测系统设置内保存的日志时长为准 )
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 **上下文** 图标即可
- 支持 导出日志

image

image

## MySQL 慢日志

MySQL 慢查询日志 ( Slow Query Log ) 是 MySQL 数据库引擎提供的一种功能，用于记录执行时间超过指定阈值的查询语句。通过记录执行时间较长的查询语句，可以帮助数据库管理员识别并优化数据库中的性能问题。MySQL 慢日志的默认阈值为 1 s，则数据服务的 MySQL 记录数据库中执行时间超过 1s (可以在 **参数设置** 中修改 `long_query_time` 参数来设置阈值) 的 SQL 语句，并进行相似语句去重。

## 开启慢日志

1. 进入 MySQL 数据库列表，点击目标 MySQL 实例名进入实例详情。
2. 点击左侧导航栏中 慢日志，并切换到 慢日志管理 页签。
3. 点击开启慢日志功能。

!!! info

开启慢日志功能会导致实例重启！

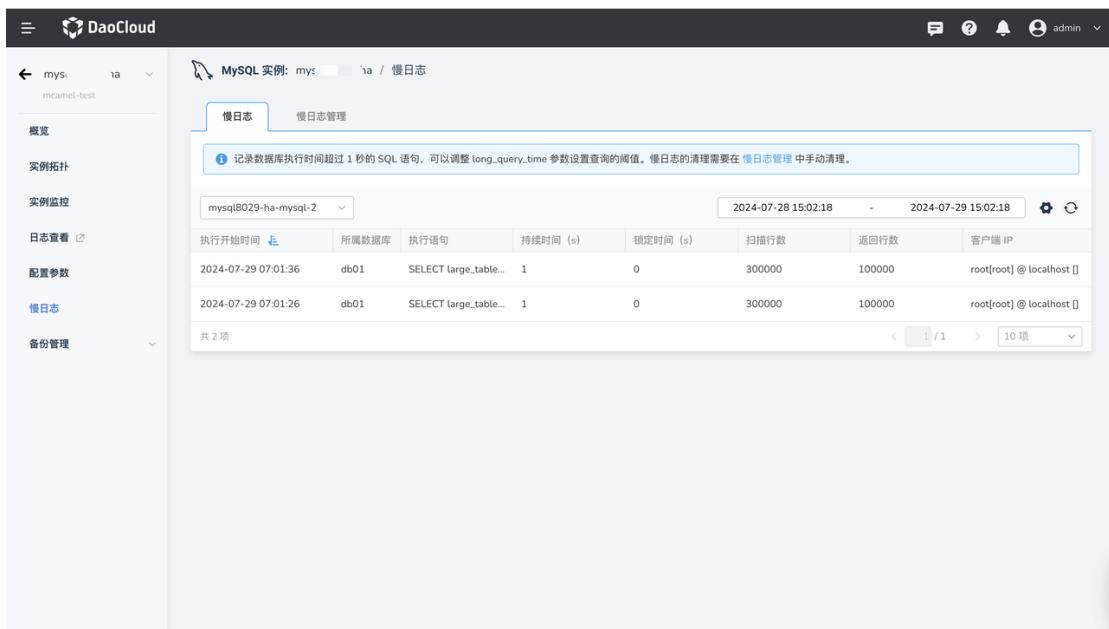


alt text

## 查看慢日志

1. 进入 MySQL 数据库列表，点击目标 MySQL 实例名进入实例详情。
2. 点击左侧导航栏中 慢日志，选择需要查看的 MySQL 容器组和时间范围。

默认不开启慢日志功能，可前往 **慢日志管理** 页签开启该功能。



alt text

## 删除慢日志

1. 点击左侧导航栏中 慢日志，并切换到 慢日志管理 页签。
2. 在清除慢日志模块，可以查看当前实例的慢日志数量。
3. 勾选想要清理的 MySQL 容器组，点击删除即可清除对应的慢日志数据。

**!!! info**

目前没有自动清理慢日志机制，需要手动执行！

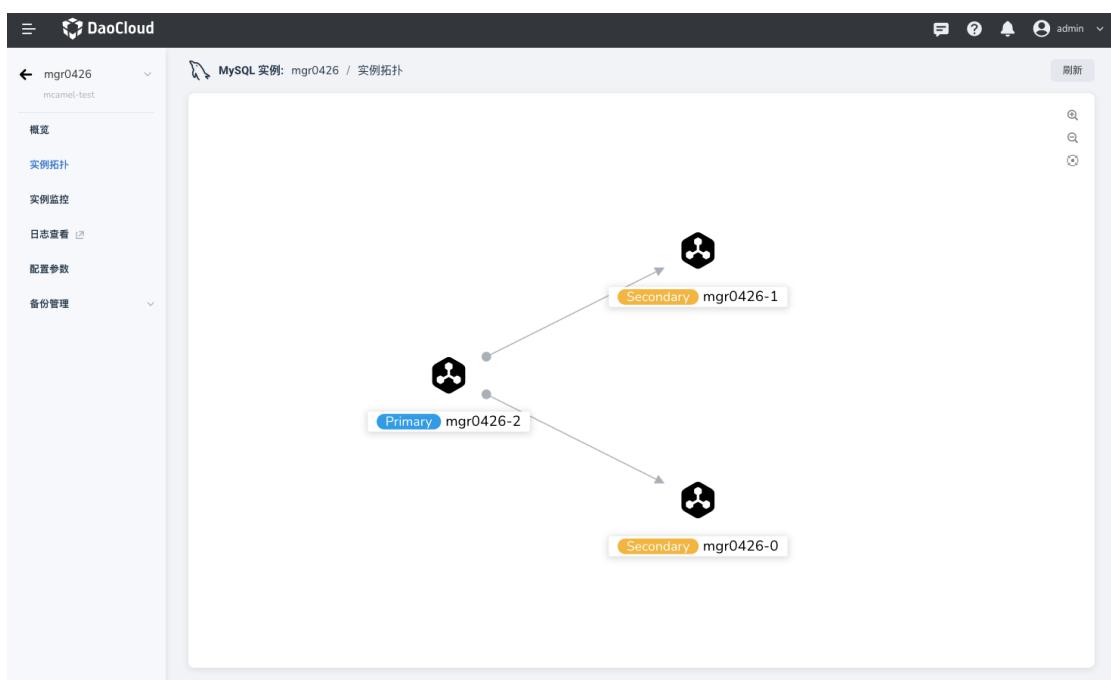
## 实例拓扑

MySQL 实例拓扑展示实例中节点间数据同步的实例状态，以及基本信息。

## 操作步骤

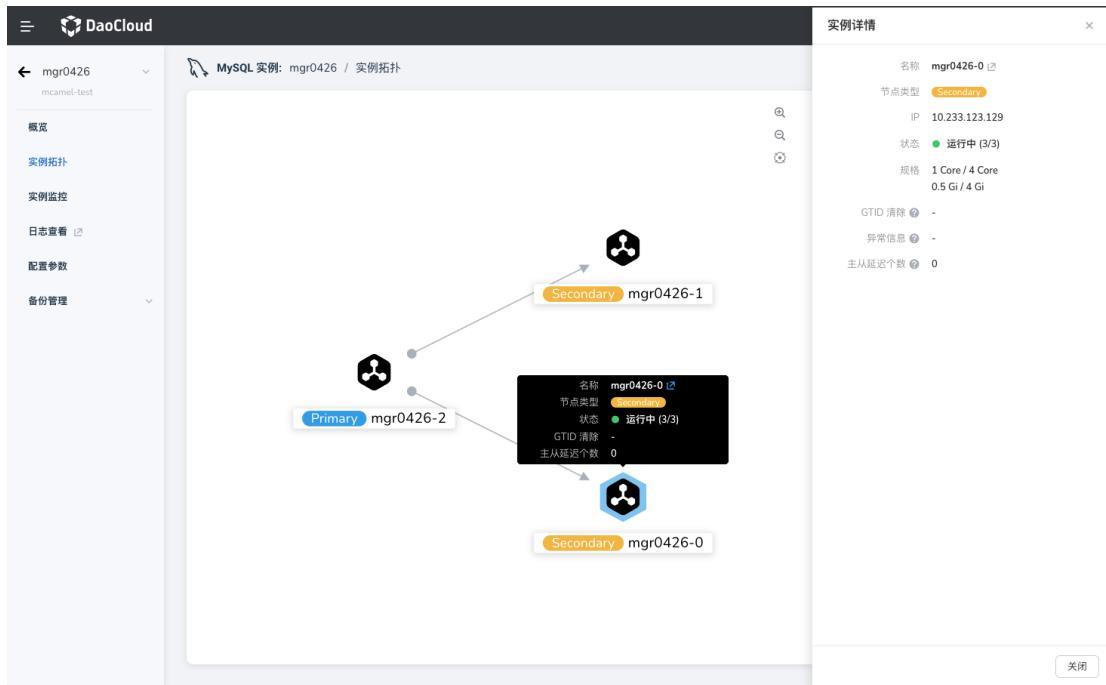
点击进入目标实例的详情页，点击左侧导航栏，选择 实例拓扑。

- 在拓扑中，可以看到实例节点之间的主从关系以及数据同步的方向；
- 点击右上角的图标可以放大或缩小拓扑图；



## 实例拓扑

- 鼠标悬浮在节点上或点击节点时，可以查看所选节点的详情。



## 实例拓扑

# 备份配置

DCE 数据服务提供的 MySQL 支持对数据库进行备份和恢复，以保证数据的安全性。在工作空间中，工作空间管理员可以统一管理 MySQL 数据备份的存储配置。

## 创建备份配置

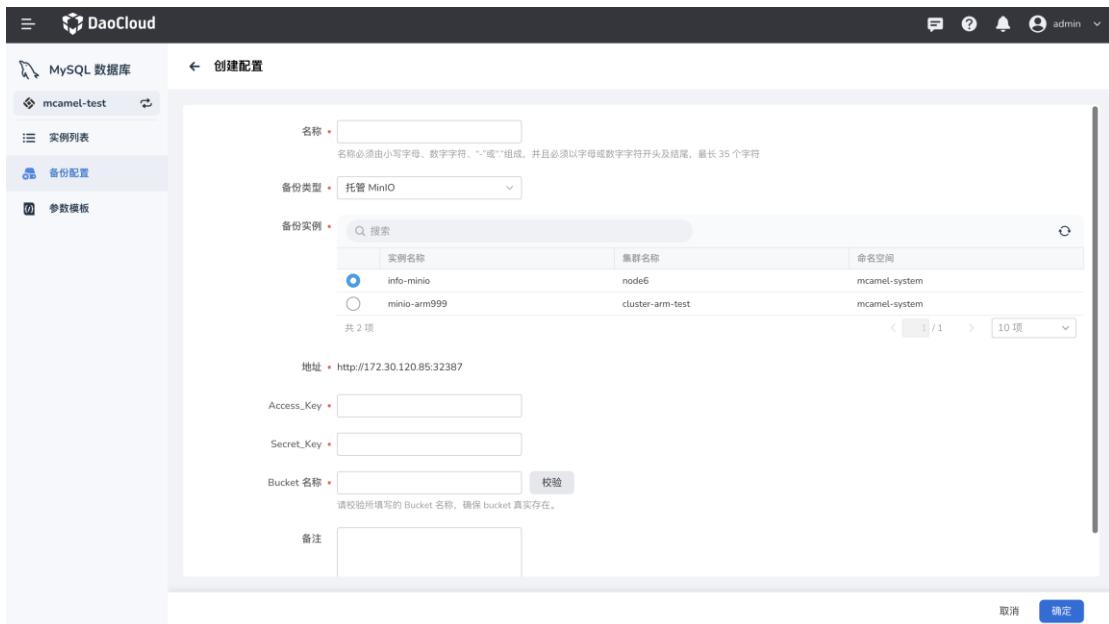
1. 点击导航栏进入 MySQL 数据库
2. 点击 MySQL 数据库导航栏中 备份配置，点击列表中的 创建 按钮

配置名称	备份类型	Access Key	备份地址	最近更新时间	创建时间
dd123	托管 MinIO	minio	http://172.30.120.85...	2024-05-27 21:19	2024-05-27 21:20
fangbin998123	托管 MinIO	minio	http://172.30.120.85...	2024-05-24 18:24	2024-05-24 18:24
s3-minio	S3	minio	http://10.6.216.22:32...	2024-05-14 09:34	2024-01-26 11:00

mysql-backup-config

### 3. 选择备份配置的类型。

- 使用 DCE 5.0 数据服务中提供的 MinIO 实例时，选中需要备份数据的 MinIO 实例，系统会自动获取所选 MinIO 的地址，用户需要填写所选 MinIO 实例的 Access\_Key、Secret\_Key、Bucket 名称。请确保 MinIO 中已存在填写的 Bucket。
- 使用其他 S3 对象存储：填写要使用的 S3 的访问地址、Access\_Key、Secret\_Key 以及 Bucket 名称。



mysql-backup-config



mysql-backup-config

4. 填写并校验完成后，点击 **确认** 后返回备份配置列表，即可查看创建成功的存储信息。

## 编辑备份配置

进入备份配置列表，点击想要编辑的对象存储最后一列的 ... -> **更新**，对其进行修改。

## 删除备份配置

进入备份配置列表，点击想要编辑的对象存储最后一列的 ... -> **删除**。

若工作空间中的中间件实例正在使用所选的对象存储实例，则不允许被删除。



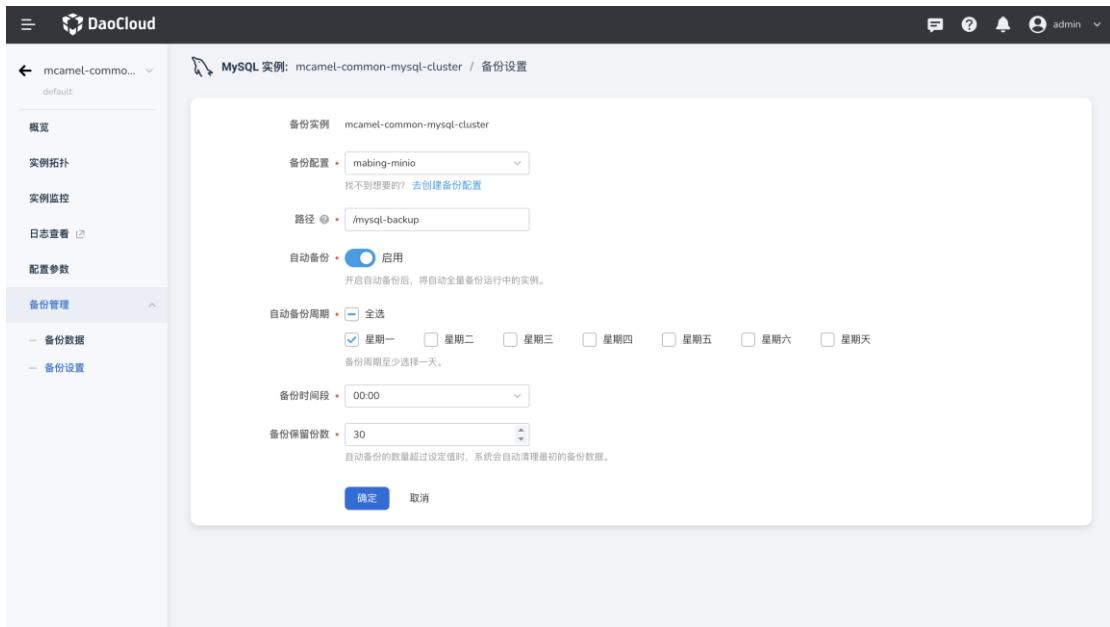
mysql-backup-config

## 配置自动备份

MySQL 数据库支持数据库实例的自动备份，由于开启备份会损耗数据库读写性能，因此建议在业务低峰时间段对其自动备份。以保证数据库存在数据丢失时能够快速找回并保证数据安全。

### 操作步骤

1. 进入 **MySQL 数据库**。
2. 在实例列表中选择需要开启自动备份的实例，点击其名称进入实例详情。
3. 点击左侧导航栏中的 **备份管理 -> 备份设置**。



### auto-backup

- 备份配置：选择工作空间中已配置的对象存储实例，将备份数据存储在该所选实例中；
- 路径：填写在对象存储中的地址，完整的 S3 路径格式为 s3://bucket-name/object-key。此处需要以 / 开头。
- 自动备份：默认不开启。若开启，则按照设置的自动备份周期和时间定时对 MySQL 实例进行备份。
- 自动备份周期：默认为全选。
  - 全选：选择一周内的每一天。系统每天都会进行自动备份。
  - 选择周期：选择一周内的一天或几天。系统会在所选时间进行自动备份。
- 备份时间段：在所选时间内自动进行备份。
- 备份保留份数：当该实例备份的数量达到设定值时，则删除超过该值的最早的数据。

# 手动备份

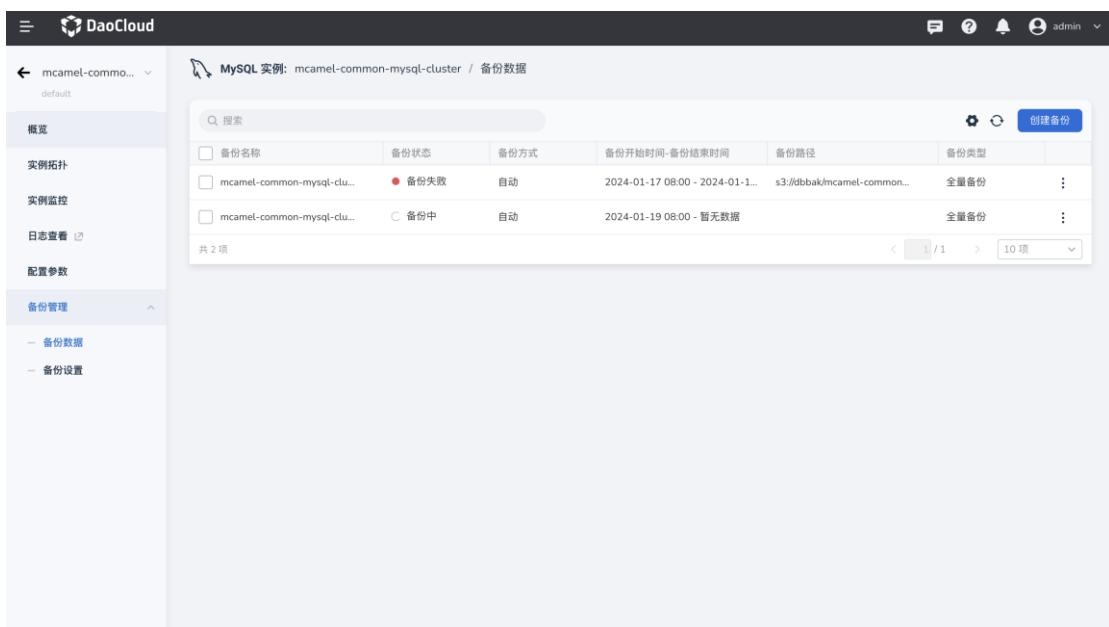
MySQL 数据库支持对运行中 状态的实例进行手动备份，以便随时对数据库数据进行备份，以保证数据安全。

## 操作步骤

### 1. 进入 MySQL 数据库

2. 在实例列表中选择需要开启自动备份的实例，点击其名称进入实例详情。

3. 点击左侧导航栏中的 备份管理 -> 备份数据。

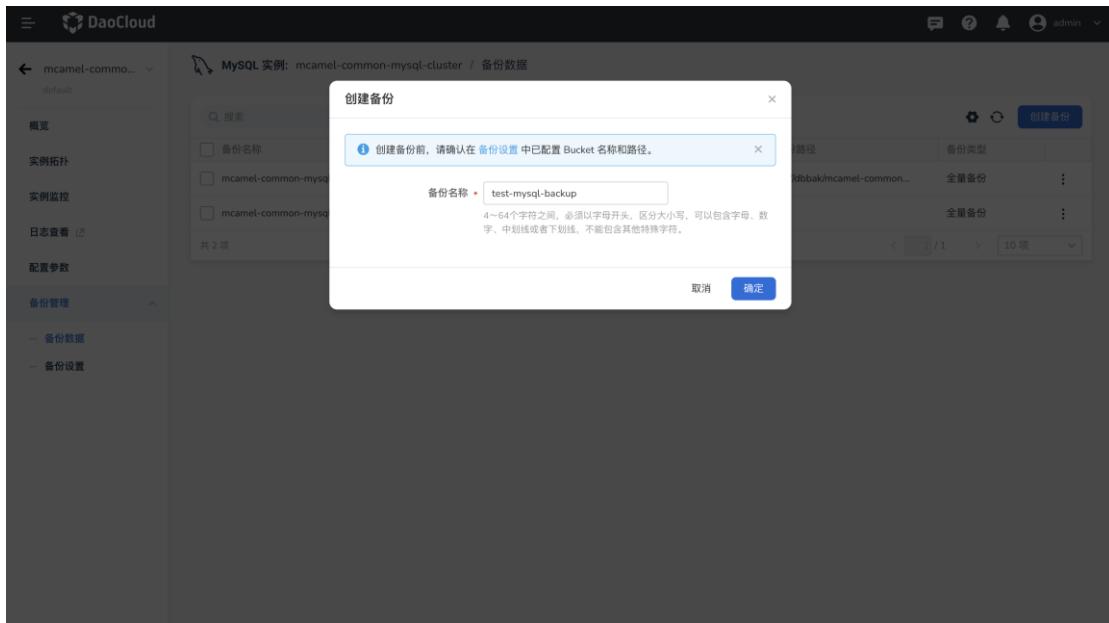


The screenshot shows the MySQL instance details page in the DaoCloud Enterprise 5.0 interface. The left sidebar has sections like Overview, Instance Topology, Instance Monitoring, and Configuration Parameters, with 'Backup Management' currently selected. Under 'Backup Management', there are two sub-options: 'Backup Data' and 'Backup Settings'. The main content area displays a table of backup data for the 'mcamel-common-mysql-cluster' instance. The table columns include Backup Name, Status, Method, Start Time-End Time, Path, and Type. One row shows a failed backup attempt ('备份失败') and another shows a backup in progress ('备份中'). A blue 'Create Backup' button is located at the top right of the table area.

manual-backup

4. 点击列表右上角的 创建备份 按钮，并填写备份的名称。

- 确保 备份管理 -> 备份设置 中已选择目标备份的对象存储实例以及存储的路径。
- 可视情况确认是否开启自动备份。



manual-backup

5. 点击 **确认**，返回备份列表，即可在列表中查看备份的状态。

## MySQL 跨集群同步

MySQL 自带的复制能力提供了主从、多从、多主、级联等多种复制方式，并支持跨版本

( 5.x , 8.x ) 数据同步。

本例以 1 对 1 主从为例，实现跨集群的 MySQL 实例间数据同步，源端与目标端均为 3

副本实例，目标端实例在同步期间仅提供只读服务，解除同步关系后，可作为独立实例运

行。

### !!! caution

复制功能仅执行增量同步，任务开始前已产生的数据差异不会同步，因此需在目标库中创建相同的库结构，

或采用 mysqldump 命令先做一个全量数据同步，下面以 mysqldump 为例。

## 全量数据 dump ( 可选 )

当用户的业务数据库具有一定存量数据时，建议先采用 dump 操作，完成一次全量备份，

确保主从库之间的库结构、内容一致。

### 1. 登录源端节点，执行备份操作：

```
进入源端 Pod
kubectl exec -it [Pod 名称] -- /bin/bash
锁表，防止数据出现不一致情况
mysql> FLUSH TABLES WITH READ LOCK;
对目标数据库执行 dump
shell> mysqldump -u [用户名] --set-gtid-purged=off -p [数据库名称] > backup.sql
对表释放锁。
mysql> UNLOCK TABLES;
退出源 Pod 终端，回到节点控制台
shell> exit
将备份文件从源 Pod 复制到所在节点上。
kubectl cp [源 Pod 名称]:/path/to/backup.sql /path/on/local/computer/backup.sql
将备份文件复制到目标节点。
scp backup.sql [用户名]@[目标节点地址]:[存放路径]
```

### 2. 登录目标端节点，执行恢复操作：

```
将备份文件从所在节点复制到目标端 Pod 上。
kubectl cp backup.sql [目标端 Pod 名称]:[存放路径] -n [命名空间]
进入目标端 Pod
kubectl exec -it [目标端 Pod 名称] -n [命名空间] -- /bin/bash
进入目标端 pod，执行数据恢复命令
shell> mysql -u [用户名] -p [数据库名称] < backup.sql
```

## 增量数据同步

在主从库库结构一致的前提下，即可采用 MySQL 提供的复制功能持续同步。操作如下：

## 源端

### 1. 进入中间件的参数配置页面，确定以下配置参数：

```
server-id = <实例的唯一标识符> #源实例与目标实例的 ID 必须不同
log-bin = <二进制日志文件的路径和名称>
binlog-format = Mixed
```

其中 **server-id** 未在参数配置页面提供，修改方法如下：

1. 进入实例的 CR 文件：容器管理 - 实例所在集群 - 自定义资源 -

[mysqlclusters.mysql.presslabs.org - 实例 CR](#)

## 2. 增加字段 : spec.serverIDOffset: 200

sync

sync

!!! note

修改参数后会重启数据库，并导致服务终端。请谨慎操作。

## 2. 创建复制账户

进入 mysql 服务的控制台，执行以下创建命令。该用户专用于集群间数据同步，出于

安全性考虑，建议创建该用户。

```
mysql> grant replication slave, replication client on *.* to 'rep'@'%' identified by '123456ab';
```

参数解释：

- **grant** : 授权
- **replication** : 授予复制权限
- **.** : 所有库和所有表
- **rep** : 授权用户
- **%** : 所有机器能够访问
- **by '123456ab'** : 该用户密码

sync

sync

## 3. 服务配置

进入 **容器管理** 模块，为实例配置一个 Nodeport 服务，用于目标端实例的同步访

问：

sync

sync

- 服务端口、容器端口：3306
- 添加标签：role/master

## 目标端

1. 确保目标端的 server-id 与源端不同，如需配置可参考源端配制方法。

server-id = <目标端集群的唯一标识符>

!!! note

修改参数后会重启数据库，并导致服务终端。请谨慎操作。

2. 在目标端的 mysql 命令行中配置源端信息：

```
mysql> CHANGE MASTER TO
 MASTER_HOST = '<源集群的 Nodeport 服务 IP 地址>',
 MASTER_PORT = <源集群的 Nodeport 节点端口>,
 MASTER_USER = '<用户名>',
 MASTER_PASSWORD = '<密码>',
 MASTER_LOG_FILE = '<源集群的 File 值>', # 源端操作中记录的 File 值
 MASTER_LOG_POS = <源集群的 Position 值>, # 源端操作中记录的 Position 值
 MASTER_RETRY_COUNT = <重试连接的次数>; # 0 为无限制
```

其中 File 与 Position 字段可在源端的控制台查看，查看命令如下：

mysql> SHOW MASTER STATUS;

sync

sync

样例：

```
mysql> CHANGE MASTER TO
 MASTER_HOST = '172.30.120.202',
 MASTER_PORT = 32470,
 MASTER_USER = 'rep',
 MASTER_PASSWORD = '123456ab',
 MASTER_LOG_FILE = 'mysql-bin.000003',
 MASTER_LOG_POS = 2595935,
 MASTER_RETRY_COUNT = 0;
```

3. 启动同步，启动后将自动持续执行两实例间的数据同步；

mysql> start slave;

4. 状态检测，重点关注以下两项，状态为 Yes，表示复制功能已开始运行。

mysql> SHOW SLAVE STATUS\G

sync

sync

!!! note

此时目标端处于 slave 状态，在中间件列表中将显示为 未就绪 状态，这是正常的，解除主从关系后可以恢复正常。

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/mysql/images/sync06.png)

## 目标端提供服务

当源端出现故障后，目标端转换角色对外提供服务，需要首先解除主从同步关系。操作如下

```
如需解除主从关系，首先需要停止从节点角色
mysql> stop slave;
mysql> reset slave all;
重启目标端的数据库服务
mysql> service mysql restart
```

重启后，目标端实例将恢复独立运行。

## 数据恢复

如需数据恢复，可通过 dump 方式实现数据恢复，可参考 **全量数据 dump** 这里不再赘述。

## MySQL 归档方案

pt-archiver 是用来归档表的工具，可以做到低影响、高性能的归档工具，从表中删除旧数据，而不会对 OLTP 查询产生太大影响。可以将数据插入到另一个表中，该表不需要在同一台服务器上。可以将其写入适合 LOAD DATA INFILE 的格式的文件中。或者两者都不做，只做删除。在归档的时候也可以指定归档的列和行。

## 安装

默认部署 MySQL 时已经安装了 **pt-heartbeat** 工具，通过以下命令检查：

```
[root@mysql1012-mysql-2 /]# pt-heartbeat --version
pt-heartbeat 3.4.0
```

**pt-heartbeat** 至少需要指定 **-dest**、**-file**、**-purge** 其中的一个，有一些选项是互斥的。

```
Specify at least one of --dest, --file, or --purge.
--ignore and --replace are mutually exclusive.
--txn-size and --commit-each are mutually exclusive.
--low-priority-insert and --delayed-insert are mutually exclusive.
--share-lock and --for-update are mutually exclusive.
--analyze and --optimize are mutually exclusive.
--no-ascend and --no-delete are mutually exclusive.
DSN values in --dest default to values from --source if COPY is yes
```

## 归档方法

### 仅删除数据，不归档

```
pt-heartbeat \
- -source h=172.30.47.0,u=root,p='ZoOll1K%YbG!zlh',P=31898,D=test,t=myTableSimple \
- -purge \
- -where "1=1" \
- -nosafe-auto-incremen
```

### 归档到文件

文件格式：通过 **-output-format** 指定，归档出来的文件有 header：使用 **-header** 选项。

- dump: MySQL dump format using tabs as field separator (default)
- csv: Dump rows using ‘‘ as separator and optionally enclosing fields by “”. This format is equivalent to FIELDS TERMINATED BY ‘‘ OPTIONALLY ENCLOSED BY “”.

恢复：可以采用 LOAD DATA local INFILE 语法

```
pt-heartbeat \
- -source h=172.30.47.0,u=root,p='ZoOll1K%YbG!zlh',P=31898,D=test,t=one_column \
- -file=/var/log/one_column.txt \
```

```

- -progress 5000 \
- -where "1=1" \
- -no-delete \
- -statistics --limit=10000 --txn-size 1000 --nosafe-auto-incremen

cat /var/log/one_column.txt
1 zhangsan

pt-archiver \
- -source h=172.30.47.0,u=root,p='ZoO1l1K%YbG!zlh',P=31898,D=test,t=one_column \
- -file=/var/log/one_column_csv.txt \
- -output-format=csv \
--progress 5000 \
- -where "1=1" \
- -no-delete \
- -statistics --limit=10000 --txn-size 1000 --nosafe-auto-incremen

cat /var/log/one_column_csv.txt
1, "zhangsan"

使用--header 选项，这里要保证--file 指定的文件不存在才会添加 header
pt-archiver \
- -source h=172.30.47.0,u=root,p='ZoO1l1K%YbG!zlh',P=31898,D=test,t=one_column \
- -file=/var/log/one_column_csv.txt \
- -output-format=csv \
- -header \
- -progress 5000 \
- -where "1=1" \
- -no-delete \
- -statistics --limit=10000 --txn-size 1000 --nosafe-auto-incremen

cat /var/log/one_column_csv.txt
id name
1 zhangsan

恢复
mysql -uroot -p'ZoO1l1K%YbG!zlh' -h172.30.47.0 -P31898 --local-infile=1
LOAD DATA local INFILE '/var/log/one_column_csv.txt' INTO TABLE one_column;

```

## 不做任何操作，只打印要执行的查询语句，**-dry-run** 选项

```

pt-archiver \
- -source h=172.30.47.0,u=root,p='ZoO1l1K%YbG!zlh',P=31898,D=test,t=myTableSimple \
- -purge \

```

```
- -where "1=1" \
- -dry-run
```

## 归档到别的表（这个表可以在另一个数据库实例）

```
pt-archiver \
- -source h=172.30.47.0,u=root,p='ZoO1l1K%YbG!zlh',P=31898,D=test,t=myTableSimple \
- -dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
- -where "1=1" \
- -no-delete
```

## 指定要归档的列，**-columns** 参数

```
pt-archiver \
- -source h=172.30.47.0,u=root,p='ZoO1l1K%YbG!zlh',P=31898,D=test,t=myTableSimple \
- -dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
- -where "1=1" \
- -no-delete \
- -columns=name,email
```

## 有从库的归档，从库延迟大于 1s 就暂停归档：-check-slave-lag

```
在 replica 里执行
mysql>stop slave;
mysql>CHANGE MASTER TO MASTER_DELAY = 10;
mysql>start slave;
mysql>show slave status \G;
```

```
pt-archiver \
- -source h=127.0.0.1,u=root,p='ZoO1l1K%YbG!zlh',P=3306,D=test,t=myTableSimple \
- -dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
- -where="1=1" \
- -max-lag=1 \
- -check-slave-lag h=10.244.2.30,u=root,p='ZoO1l1K%YbG!zlh',P=3306,D=test \
- -check-interval 1s \
- -progress=1 \
- -statistics
```

## 常见问题

1. 归档出来的新数据总是少一行，可参考[故障分析 | pt-archiver 归档丢失一条记录](#)

# 加上--dry-run 查看生成的语句，注意 WHERE (1=1) AND ( \_\_id\_\_ < '3')

```
pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTableSimple \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
--where "1=1" \
--no-delete \
--dry-run

SELECT /*!40001 SQL_NO_CACHE */ __id__ , __name__ , __phone__ , __email__ ,
__address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__ , __num
berrange__ , __currency__ , __alphanumeric__ FROM __test__ . __myTableSimple__ F
ORCE INDEX(__PRIMARY__) WHERE (1=1) AND (__id__ < '3') ORDER BY __
_id__ LIMIT 1
SELECT /*!40001 SQL_NO_CACHE */ __id__ , __name__ , __phone__ , __email__ ,
__address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__ , __num
berrange__ , __currency__ , __alphanumeric__ FROM __test__ . __myTableSimple__ F
ORCE INDEX(__PRIMARY__) WHERE (1=1) AND (__id__ < '3') AND ((__id__
> ?)) ORDER BY __id__ LIMIT 1
INSERT INTO __test__ . __myTableSimple__ (__id__ , __name__ , __phone__ , __e
mail__ , __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__
, __numberrange__ , __currency__ , __alphanumeric__) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

```
SELECT MAX(id) from myTableSimple;
+-----+
| MAX(id) |
+-----+
| 3|
+-----+
```

### 解决办法一：--nosafe-auto-incremen

```
pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTableSimple \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
--where "1=1" \
--no-delete \
--nosafe-auto-incremen \
--dry-run
```

```

SELECT /*!40001 SQL_NO_CACHE */ _id__ , __name__ , __phone__ , __email__ ,
 __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__ , __num
berrange__ , __currency__ , __alphanumeric__ FROM __test__ . __myTableSimple__ F
ORCE INDEX(__PRIMARY__) WHERE (1=1) ORDER BY __id__ LIMIT 1
SELECT /*!40001 SQL_NO_CACHE */ _id__ , __name__ , __phone__ , __email__ ,
 __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__ , __num
berrange__ , __currency__ , __alphanumeric__ FROM __test__ . __myTableSimple__ F
ORCE INDEX(__PRIMARY__) WHERE (1=1) AND ((__id__ > ?)) ORDER BY __
_id__ LIMIT 1
INSERT INTO __test__ . __myTableSimple__ (__id__ , __name__ , __phone__ , __e
mail__ , __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__
, __numberrange__ , __currency__ , __alphanumeric__) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
?
```

## 解决办法二：-no-ascend 和在-source 的 DSN 里通过 i=specified\_index 指定索引

```

pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTableSimple,i=
name_index \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
--where "1=1" \
--no-delete \
--dry-run

```

```

SELECT /*!40001 SQL_NO_CACHE */ _id__ , __name__ , __phone__ , __email__ ,
 __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__ , __num
berrange__ , __currency__ , __alphanumeric__ FROM __test__ . __myTableSimple__ F
ORCE INDEX(__name_index__) WHERE (1=1) ORDER BY __name__ LIMIT 1
SELECT /*!40001 SQL_NO_CACHE */ _id__ , __name__ , __phone__ , __email__ ,
 __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__ , __num
berrange__ , __currency__ , __alphanumeric__ FROM __test__ . __myTableSimple__ F
ORCE INDEX(__name_index__) WHERE (1=1) AND (((? IS NULL AND __name_
__ IS NOT NULL) OR (__name__ > ?))) ORDER BY __name__ LIMIT 1
INSERT INTO __test__ . __myTableSimple__ (__id__ , __name__ , __phone__ , __e
mail__ , __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__
, __numberrange__ , __currency__ , __alphanumeric__) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
?
```

## 2. 没有主键，采用默认参数会归档失败

- 默认会去找 **ascendable index**，如果没有就会失败。
- 可以在 **-source** 的 DSN 指定其他索引：**i=specified\_index**

```

show create table myTableNoPrimaryKey\G

1. row *****

Table: myTableNoPrimaryKey
Create Table: CREATE TABLE __myTableNoPrimaryKey__ (
 __id__ mediumint NOT NULL,
 __name__ varchar(255) DEFAULT NULL,
 __phone__ varchar(100) DEFAULT NULL,
 __email__ varchar(255) DEFAULT NULL,
 __address__ varchar(255) DEFAULT NULL,
 __list__ varchar(255) DEFAULT NULL,
 __country__ varchar(100) DEFAULT NULL,
 __region__ varchar(50) DEFAULT NULL,
 __postalZip__ varchar(10) DEFAULT NULL,
 __text__ text,
 __numberrange__ mediumint DEFAULT NULL,
 __currency__ varchar(100) DEFAULT NULL,
 __alphanumeric__ varchar(255) DEFAULT NULL,
 KEY __name_index__ (__name__)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.03 sec)

```

# 没有主键，没有在--source 里指定索引，失败

```

pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTableNoPrimar
yKey \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
--where "1=1"

```

Cannot find an ascendable index in table at /usr/bin/pt-archiver line 3261.

解决办法：在--source 的 DSN 里通过 i=other\_index 指定其他索引

```

pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTableNoPrimar
yKey,i=name_index \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
--where "1=1"

```

### 解决办法：在--source 的 DSN 里通过 i=other\_index 指定其他索引

```

pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTableNoPrimar
yKey,i=name_index \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507 \
--where "1=1"

```

### 3. 批量插入失败

- 当使用了--bulk-insert 的时候，出现失败的情况。

```
pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTable
Simple \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507,D=test,t=myTableSimple \
--where "1=1" \
--bulk-insert \
--limit=1000 --no-delete --progress 10 --statistics
TIME ELAPSED COUNT
2023-10-16T10:37:32 0 0
DBD::mysql::st execute failed: Loading local data is disabled; this must be enabled on both the client and server sides [for Statement "LOAD DATA LOCAL INFILE ? INTO TABLE __test__ . __myTableSimple__ (__id__ , __name__ , __phone__ , __email__ , __address__ , __list__ , __country__ , __region__ , __postalzip__ , __text__ , __numberrange__ , __currency__ , __alphanumeric__)" with ParamValues: 0='/tmp/GPJHnHSRUspt-archiver'] at /usr/bin/pt-archiver line 6876.
```

- 查看 MySQL 相关变量

```
mysql> show variables like 'local_infile';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| local_infile | OFF |
+-----+-----+
1 row in set (0.04 sec)
```

- 需要在 client 和 server 两边都设置 local\_infile=on, set global local\_infile=on;

还需要在--source 和--dest 里设置 L=1

```
需要在 source 和 dest 两边的 MySQL 都设置
mysql> set global local_infile=on;
Query OK, 0 rows affected (0.04 sec)
```

```
pt-archiver \
--source h=172.30.47.0,u=root,p='ZoO111K%YbG!zlh',P=31898,D=test,t=myTable
Simple,L=1 \
--dest h=172.30.47.0,u=root,p='12345678@',P=31507,D=test,t=myTableSimple \
--where "1=1" \
--bulk-insert \
--limit=1000 --no-delete --progress 10 --statistics
```

TIME	ELAPSED	COUNT	
2023-10-16T10:46:28	0	0	
2023-10-16T10:46:28	0	2	
Started at 2023-10-16T10:46:28, ended at 2023-10-16T10:46:29			
Source: D=test,L=1,P=31898,h=172.30.47.0,p=...,t=myTableSimple,u=root			
Dest: D=test,L=1,P=31507,h=172.30.47.0,p=...,t=myTableSimple,u=root			
SELECT 2			
INSERT 2			
DELETE 0			
Action	Count	Time	Pct
commit	6	0.2151	45.16
bulk_inserting	1	0.1418	29.77
select	2	0.0763	16.02
print_bulkfile	2	0.0000	0.00
other	0	0.0431	9.04

## MySQL 健康检查

如果您发现遇到的问题，未包含在 [故障排查](#) 内，可以快速跳转到页面底部，提交您的问题。

常规的 MySQL 健康状态检查，可以通过一句命令快速查看 MySQL 实例的整体状态：

```
kubectl get pod -n mcamel-system -Lhealthy,role | grep mysql
```

输出类似于：

mcamel-common-mysql-cluster-auto-2023-03-28t00-00-00-backujgg9m	0/1	Completed
0	27h	
mcamel-common-mysql-cluster-auto-2023-03-29t00-00-00-backusgf59	0/1	Completed
0	3h43m	
mcamel-common-mysql-cluster-mysql-0	4/4	Running
6 (11h ago)	25h	yes
mcamel-common-mysql-cluster-mysql-1	4/4	Running
690 (11h ago)	4d20h	yes
mcamel-mysql-apiserver-9797c7f76-bvf5n	2/2	Running
0	22h	
mcamel-mysql-ui-7ffd9dd8db-d5jfm	2/2	Running
0	25m	
mysql-operator-0	2/2	Running
109 (47m ago)	2d21h	

如上所示，如果主备节点（`master` 和 `replica`）的状态均为 `yes`，说明 MySQL 为正常

状态。

## MySQL Pod

可以通过以下命令快速的查看当前集群上所有 MySQL 实例的健康状态：

```
kubectl get mysql -A
```

输出类似于：

NAMESPACE	NAME	READY	REPLICAS	AGE
ghippo-system	test	True	1	3d
mcamel-system	mcamel-common-mysql-cluster	False	2	62d

针对不同的副本状态，排障方案如下文所述。

## Pod running = 0/4 , 状态为 Init: Error

遇到此类问题时，首先应该查看 master 节点的（sidecar）日志信息。

```
kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep master | awk '{print $1}' | xargs -I {} kubectl logs -f {} -n mcamel-system -c sidecar
```

??? note “输出内容示例”

```
```none
2023-02-09T05:38:56.208445-00:00 0 [Note] [MY-011825] [Xtrabackup] perl binary not found.
Skipping the version check
2023-02-09T05:38:56.208521-00:00 0 [Note] [MY-011825] [Xtrabackup] Connecting to MySQL
server host: 127.0.0.1, user: sys_replication, password: set, port: not set, socket: not set
2023-02-09T05:38:56.212595-00:00 0 [Note] [MY-011825] [Xtrabackup] Using server version 8.
0.29
2023-02-09T05:38:56.217325-00:00 0 [Note] [MY-011825] [Xtrabackup] Executing LOCK INST
ANCE FOR BACKUP ...
2023-02-09T05:38:56.219880-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Found tables with r
ow versions due to INSTANT ADD/DROP columns
2023-02-09T05:38:56.219931-00:00 0 [ERROR] [MY-011825] [Xtrabackup] This feature is not s
table and will cause backup corruption.
2023-02-09T05:38:56.219940-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Please check https://
docs.percona.com/percona-xtrabackup/8.0/em/instant.html for more details.
2023-02-09T05:38:56.219945-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Tables found:
2023-02-09T05:38:56.219951-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/USER_SES
SION
```

2023-02-09T05:38:56.219956-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/AUTHENTICATION_EXECUTION

2023-02-09T05:38:56.219960-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/AUTHENTICATION_FLOW

2023-02-09T05:38:56.219968-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/AUTHENTICATION_CONFIG

2023-02-09T05:38:56.219984-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/CLIENT_SESSION

2023-02-09T05:38:56.219991-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/IDENTITY_PROVIDER

2023-02-09T05:38:56.219998-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/PROTOCOL_MAPPER

2023-02-09T05:38:56.220005-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/RESOURCE_SERVER_SCOPE

2023-02-09T05:38:56.220012-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/REQUIRED_ACTION_PROVIDER

2023-02-09T05:38:56.220018-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/COMPONENT

2023-02-09T05:38:56.220027-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/RESOURCE_SERVER

2023-02-09T05:38:56.220036-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/CREDENTIALIAL

2023-02-09T05:38:56.220043-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/FED_USER_CREDENTIALIAL

2023-02-09T05:38:56.220049-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/MIGRATION_MODEL

2023-02-09T05:38:56.220054-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/REALM

2023-02-09T05:38:56.220062-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/CLIENT

2023-02-09T05:38:56.220069-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/REALM_ATTRIBUTE

2023-02-09T05:38:56.220075-00:00 0 [ERROR] [MY-011825] [Xtrabackup] keycloak/OFFLINE_USER_SESSION

2023-02-09T05:38:56.220084-00:00 0 [ERROR] [MY-011825] [Xtrabackup] Please run OPTIMIZE TABLE or ALTER TABLE ALGORITHM=COPY on all listed tables to fix this issue.

E0209 05:38:56.223635 1 deleg.go:144] sidecar "msg"="failed waiting for xtrabackup to finish" "error"="exit status 1"

```

登录 master 节点的 MySQL , 执行 alter 表结构 :

```
[root@master-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep master
mcamel-common-mysql-cluster-mysql-0
```

#获取密码

```
[root@master-01 ~]$ kubectl get secret -n mcamel-system mcamel-common-mysql-cluster-secret
-o=jsonpath='{.data.ROOT_PASSWORD}' | base64 -d

[root@master-01 ~]$ kubectl exec -it mcamel-common-mysql-cluster-mysql-0 -n mcamel-system
-c mysql -- /bin/bash

注意: 修改表结构需要root权限登录
~bash: mysql -uroot -p

??? note "SQL语句如下"

```sql
use keycloak;
ALTER TABLE USER_SESSION ALGORITHM=COPY;
ALTER TABLE AUTHENTICATION_EXECUTION ALGORITHM=COPY;
ALTER TABLE AUTHENTICATION_FLOW ALGORITHM=COPY;
ALTER TABLE AUTHENTICATOR_CONFIG ALGORITHM=COPY;
ALTER TABLE CLIENT_SESSION ALGORITHM=COPY;
ALTER TABLE IDENTITY_PROVIDER ALGORITHM=COPY;
ALTER TABLE PROTOCOL_MAPPER ALGORITHM=COPY;
ALTER TABLE RESOURCE_SERVER_SCOPE ALGORITHM=COPY;
ALTER TABLE REQUIRED_ACTION_PROVIDER ALGORITHM=COPY;
ALTER TABLE COMPONENT ALGORITHM=COPY;
ALTER TABLE RESOURCE_SERVER ALGORITHM=COPY;
ALTER TABLE CREDENTIAL ALGORITHM=COPY;
ALTER TABLE FED_USER_CREDENTIAL ALGORITHM=COPY;
ALTER TABLE MIGRATION_MODEL ALGORITHM=COPY;
ALTER TABLE REALM ALGORITHM=COPY;
ALTER TABLE CLIENT ALGORITHM=COPY;
ALTER TABLE REALM_ATTRIBUTE ALGORITHM=COPY;
ALTER TABLE OFFLINE_USER_SESSION ALGORITHM=COPY
```
```

```

Pod running = 2/4

此类问题很可能是因为 MySQL 实例使用的磁盘用量达到了 100%，可以在 master 节点上运行以下命令检测磁盘用量。

```
kubectl get pod -n mcamel-system | grep cluster-mysql | awk '{print $1}' | xargs -I {} kubectl
exec {} -n mcamel-system -c sidecar -- df -h | grep /var/lib/mysql
```

输出类似于：

/dev/drbd43001	50G	30G	21G	60%	/var/lib/mysql
/dev/drbd43005	80G	29G	52G	36%	/var/lib/mysql

如果发现某个 PVC 满了，则需要为 PVC 扩容。

```
kubectl edit pvc data-mcamel-common-mysql-cluster-mysql-0 -n mcamel-system # 修改 request  
大小即可
```

Pod running = 3/4

image

使用 `kubectl describe` 上图中框起来的 Pod，发现异常提示：`Warning Unhealthy 4m50s (x7194 over 3h58m) kubelet Readiness probe failed:`

此时需要手工进行修复，这是目前开源 `mysql-operator` 版本的 [Bug](#)

修复方式有两种：

- 重启 `mysql-operator`，或者
- 手工更新 `sys_operator` 的配置状态

```
kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'update sys_operator.status set value="1" WHERE name="configured"'
```

MySQL Operator

未指定 storageClass

由于没有指定 `storageClass`，导致 `mysql-operator` 无法获取 PVC 而处于 `pending` 状态。

如果采用 Helm 启动，可以做如下设置：

1. 关闭 PVC 的申请

```
orchestrator.persistence.enabled=false
```

2. 指定 storageClass 去获取 PVC

```
orchestrator.persistence.storageClass={storageClassName}
```

如果使用其他工具，可以修改 `value.yaml` 内对应字段，即可达到和 Helm 启动一样的效果。

MySQL 主备关系

MySQL 的主备关系故障相对比较复杂，基于不同现象，会有不同的解决方案。

1. 执行以下命令确认 MySQL 状态：

```
kubectl get mysql -A
```

输出类似于：

NAMESPACE	NAME	READY	REPLICAS	AGE
ghippo-system	test	True	1	3d
mcamel-system	mcamel-common-mysql-cluster	False	2	62d

2. 关注 Ready 字段值为 False 的库（这里为 True 的判断是延迟小于 30s 同步），查看

MySQL 从库的日志

```
kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}' | xargs -I {} kubectl logs {} -n mcamel-system -c mysql | grep ERROR
```

当实例状态为 False 时，可能存在以下几类故障，可以结合库日志信息排查修复。

实例状态为 false 但日志无报错信息

如果从库的日志中没有任何错误 ERROR 信息，说明 False 只是因为主从同步的延迟过大，

可对从库执行以下命令进一步排查：

1. 寻找到从节点的 Pod

```
kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}'
```

输出类似于：

```
mcamel-common-mysql-cluster-mysql-1
```

2. 设置 binlog 参数

```
kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'set global sync_binlog=10086;'
```

3. 进入 MySQL 的容器

```
kubectl exec -it mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql --defaults-file=/etc/mysql/client.conf
```

4. 在 MySQL 容器中执行查看命令，获取从库状态。

Seconds_Behind_Master 字段为主从延迟，如果取值在 0~30，可以认为没有延迟；表示主从可以保持同步。

```
mysql> show slave status\G;
***** 1. row *****
Slave_IO_State: Waiting for source to send event
Master_Host: mcamel-common-mysql-cluster-mysql-0.mysql.mcamel
-system
Master_User: sys_replication
Master_Port: 3306
Connect_Retry: 1
Master_Log_File: mysql-bin.000304
Read_Master_Log_Pos: 83592007
Relay_Log_File: mcamel-common-mysql-cluster-mysql-1-relay-bin.0000
02
Relay_Log_Pos: 83564355
Relay_Master_Log_File: mysql-bin.000304
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 83564299
Relay_Log_Space: 83592303
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
```

```

        Last_IO_Error: 0
        Last_SQL_Error:
        Last_SQL_Errorno: 0
        Last_SQL_Error:
Replicate_Ignore_Server_Ids:
        Master_Server_Id: 100
        Master_UUID: e17dae09-8da0-11ed-9104-c2f9484728fd
        Master_Info_File: mysql.slave_master_info
        SQL_Delay: 0
        SQL_Remaining_Delay: NULL
Slave_SQL_Running_State: Replica has read all relay log; waiting for more update
dates
        Master_Retry_Count: 86400
        Master_Bind:
        Last_IO_Error_Timestamp:
        Last_SQL_Error_Timestamp:
        Master_SSL_Crl:
        Master_SSL_Crlpath:
Retrieved_Gtid_Set: e17dae09-8da0-11ed-9104-c2f9484728fd:21614244-2162
1569
        Executed_Gtid_Set: 4bc2107c-819a-11ed-bf23-22be07e4eaff:1-342297,
7cc717ea-7c1b-11ed-b59d-c2ba3f807d12:1-619197,
a5ab763a-7c1b-11ed-b5ca-522707642ace:1-178069,
a6045297-8743-11ed-8712-8e52c3ace534:1-4073131,
a95cf9df-84d7-11ed-8362-5e8a1c335253:1-493942,
b5175b1b-a2ac-11ed-b0c6-d6fbe05d7579:1-3754703,
c4dc2b14-9ed9-11ed-ac61-36da81109699:1-945884,
e17dae09-8da0-11ed-9104-c2f9484728fd:1-21621569
        Auto_Position: 1
        Replicate_Rewrite_DB:
        Channel_Name:
        Master_TLS_Version:
Master_public_key_path:
        Get_master_public_key: 0
        Network_Namespace:
1 row in set, 1 warning (0.00 sec)

```

5. 主从同步后 Seconds_Behind_Master 小于 30s , 设置 sync_binlog=1

```
kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-system -c mysql -- mysql
--defaults-file=/etc/mysql/client.conf -NB -e 'set global sync_binlog=1';
```

6. 如果此时依然不见缓解 , 可以查看从库的宿主机负载或者 IO 是否太高 , 执行以下命令 :

```
[root@master-01 ~]$ uptime
11:18 up 1 day, 17:49, 2 users, load averages: 9.33 7.08 6.28
```

load averages 在正常情况下 3 个数值都不应长期超过 10；如果超过 30 以上，请合理调配下该节点的 Pod 和磁盘。

从库日志出现 复制错误

如果从库 Pod 日志中出现从库复制错误，可能由多种原因引起，下文将针对不同情况介绍判断及修复方法。

purged binlog 错误

注意以下示例，如果出现关键字 **purged binlog**，通常需要对从库执行重建处理。

```
[root@demo-alpha-master-01 /]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}' | xargs -I {} kubectl logs {} -n mcamel-system -c mysql | grep ERROR
```

2023-02-08T18:43:21.991730Z 116 [ERROR] [MY-01057] [Repl] Error reading packet from server for channel ": Cannot replicate because the master purged required binary logs. Replicate the missing transactions from elsewhere, or provision a new slave from backup. Consider increasing the master's binary log expiration period. The GTID sets and the missing purged transactions are too long to print in this message. For more information, please see the master's error log or the manual for GTID_SUBTRACT (server_errno=1236)

2023-02-08T18:43:21.991777Z 116 [ERROR] [MY-013114] [Repl] Slave I/O for channel ": Got fatal error 1236 from master when reading data from binary log: 'Cannot replicate because the master purged required binary logs. Replicate the missing transactions from elsewhere, or provision a new slave from backup. Consider increasing the master's binary log expiration period.

The GTID sets and the missing purged transactions are too long to print in this message. For more information, please see the master's error log or the manual for GTID_SUBTRACT', Error_code: MY-013114

重建操作如下：

1. 寻找从节点的 Pod

```
[root@master-01 ~]$ kubectl get pod -n mccamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}'
mcamel-common-mysql-cluster-mysql-1
```

2. 寻找从节点的 PVC

```
[root@master-01 /]$ kubectl get pvc -n mcamel-system | grep mcamel-common-mysql-cluster-mysql-1
data-mcamel-common-mysql-cluster-mysql-1
  Bound      pvc-5840569e-834f-4236-a5c6-878e41c55c85   50Gi          RWO
    local-path           33d
```

3. 删除从节点的 PVC

```
[root@master-01 /]$ kubectl delete pvc data-mcamel-common-mysql-cluster-mysql-1 -n mc
  mel-system
persistentvolumeclaim "data-mcamel-common-mysql-cluster-mysql-1" deleted
```

4. 删除从库的 Pod

```
[root@master-01 /]$ kubectl delete pod mcamel-common-mysql-cluster-mysql-1 -n mcamel-s
  ystem
pod "mcamel-common-mysql-cluster-mysql-1" deleted
```

主键冲突错误

运行命令时出现以下错误提示：

```
[root@demo-alpha-master-01 /]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-
mysql | grep replica | awk '{print $1}' | xargs -I {} kubectl logs {} -n mcamel-system -c mys
ql | grep ERROR
2023-02-08T18:43:21.991730Z 116 [ERROR] [MY-010557] [Repl] Could not execute Write_rows event on table dr_brower_db.dr_user_info; Duplicate entry '24' for key 'PRIMARY', Error_code: 1062; handler error HA_ERR_FOUND_DUPP_KEY; the event's master logmysql-bin.000010, end_log_pos 5295916
```

在错误日志中看到 Duplicate entry '24' for key 'PRIMARY', Error_code: 1062; handler error HA_ERR_FOUND_DUPP_KEY;

这说明出现了主键冲突，或者主键不存在的错误。此时，可以以幂等模式恢复或插入空事务的形式跳过错误：

方法 1：幂等模式恢复

1. 寻找到从节点的 Pod

```
[root@master-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role | grep cluster-mysql
| grep replica | awk '{print $1}'
mcamel-common-mysql-cluster-mysql-1
```

2. 设置 mysql 幂等模式

```
[root@master-01 ~]$ kubectl exec mccamel-common-mysql-cluster-mysql-1 -n mccamel-system -c mysql --defaults-file=/etc/mysql/client.conf -NB -e 'stop slave;set global slave_exec_mode="IDEMPOTENT";set global sync_binlog=10086;start slave;'
```

方法 2：插入空事务跳过错误

```
mysql> stop slave;
mysql> SET @@SESSION.GTID_NEXT= 'xxxxx: 105220'; /* 具体数值，在日志里面提到 */
mysql> BEGIN;
mysql> COMMIT;
mysql> SET SESSION GTID_NEXT = AUTOMATIC;
mysql> START SLAVE;
```

执行完成以上操作后，观察从库重建的进度：

```
# 进入 mysql 的容器
[root@master-01 ~]$ kubectl exec -it mccamel-common-mysql-cluster-mysql-1 -n mccamel-system -c mysql --defaults-file=/etc/mysql/client.conf
```

执行以下命令，查看从库的主从延迟状态字段 **Seconds_Behind_Master**，如果取值在

0~30，表示已没有主从延迟，主库和从库基本保持同步。

```
mysql> show slave status\G;
```

确认主从同步后 (**Seconds_Behind_Master** 小于 30s)，执行以下命令，设定 MySQL 严格模

式：

```
[root@master-01 ~]$ kubectl exec mccamel-common-mysql-cluster-mysql-1 -n mccamel-system -c mysql --defaults-file=/etc/mysql/client.conf -NB -e 'stop slave;set global slave_exec_mode="STRICT";set global sync_binlog=10086;start slave;'
```

主从库复制错误

当从库出现类似 [Note] Slave: MTS group recovery relay log info based on Worker-Id 0, group_r 的错误信息，可以执行如下操作：

1. 寻找到从节点的 Pod

```
[root@master-01 ~]# kubectl get pod -n mccamel-system -Lhealthy,role | grep cluster-mysql | grep replica | awk '{print $1}'
```

```
mccamel-common-mysql-cluster-mysql-1
```

2. 设置让从库跳过这个日志继续复制

```
[root@master-01 ~]# kubectl exec mcamel-common-mysql-cluster-mysql-1 -n mcamel-syste
m -c mysql -- mysql --defaults-file=/etc/mysql/client.conf -NB -e 'stop slave;reset slave;
change master to MASTER_AUTO_POSITION = 1;start slave;';
```

!!! tip

1. 这种情况可以以幂等模式执行
2. 此种类型错误也可以重做从库

主备 Pod 均为 replica

1. 通过以下命令，发现两个 MySQL 的 Pod 均为 **replica** 角色，需修正其中一个为

master。

```
[root@aster-01 ~]$ kubectl get pod -n mcamel-system -Lhealthy,role|grep mysql
mcamel-common-mysql-cluster-mysql-0                               4/4      Running
5 (16h ago)   16h    no      replica
mcamel-common-mysql-cluster-mysql-1                               4/4      Running
6 (16h ago)   16h    no      replica
mysql-operator-0                                                 2/2      Running
1 (16h ago)   16h
```

2. 进入 MySQL 查看：

```
kubectl exec -it mcamel-common-mysql-cluster-mysql-0 -n mcamel-system -c mysql -- mys
ql --defaults-file=/etc/mysql/client.conf
```

3. 查看 **slave** 的状态信息，查询结果为空的就是原来的 **master**，如下方示例中

mysql-0 对应的内容：

```
-- mysql-0
mysql> show slave status\G;
empty set, 1 warning (0.00 sec)

-- mysql-1
mysql> show slave status\G;
***** 1. row *****
Slave_IO_State: Waiting for source to send event
Master_Host: mcamel-common-mysql-cluster-mysql-0.mysql.mcamel
-system
Master_User: sys_replication
Master_Port: 3306
Connect_Retry: 1
Master_Log_File: mysql-bin.000004
Read_Master_Log_Pos: 38164242
Relay_Log_File: mcamel-common-mysql-cluster-mysql-1-relay-bin.0000
```

02

```
Relay_Log_Pos: 38164418
Relay_Master_Log_File: mysql-bin.000004
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
Replicate_Do_DB:
Replicate_Ignore_DB:
Replicate_Do_Table:
Replicate_Ignore_Table:
Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
Last_Error:
Skip_Counter: 0
Exec_Master_Log_Pos: 38164242
Relay_Log_Space: 38164658
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
Last_IO_Error:
Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 100
Master_UUID: c16da70b-ad12-11ed-8084-0a580a810256
Master_Info_File: mysql.slave_master_info
SQL_Delay: 0
SQL_Remaining_Delay: NULL
Slave_SQL_Running_State: Replica has read all relay log; waiting for more up
dates
Master_Retry_Count: 86400
Master_Bind:
Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp:
Master_SSL_Crl:
```

```

Master_SSL_Crlpath:
Retrieved_Gtid_Set: c16da70b-ad12-11ed-8084-0a580a810256:537-59096
Executed_Gtid_Set: c16da70b-ad12-11ed-8084-0a580a810256:1-59096
Auto_Position: 1
Replicate_Rewrite_DB:
    Channel_Name:
        Master_TLS_Version:
Master_public_key_path:
Get_master_public_key: 0
Network_Namespace:
1 row in set, 1 warning (0.01 sec)

```

4. 针对 master 的 mysql shell 执行重置操作：

```
mysql > stop slave;reset slave;
```

5. 此时再手动编辑 master 的 Pod : **role replica => master ,healthy no => yes** 。

6. 针对 slave 的 mysql shell 执行：

```
mysql > start slave;
```

7. 如果主从没有建立联系，在 slave 的 mysql shell 执行：

```
-- 注意替换下 {master-host-pod-index}
```

```
mysql > change master to master_host='mcamel-common-mysql-cluster-mysql-{master-h
ost-pod-index}.mysql.mcamel-system',master_port=3306,master_user='root',master_password
='{password}',master_auto_position=1,MASTER_HEARTBEAT_PERIOD=2,MASTER_CO
NNECT_RETRY=1, MASTER_RETRY_COUNT=86400;
```

主备数据不一致

当主从实例数据不一致时，可以执行以下命令完成主从一致性同步：

```
pt-table-sync --execute --charset=utf8 --ignore-databases=mysql,sys,percona --databases=amamba
,audit,hippo,insight,ipavo,keycloak,kpanda,skoala dsn=u=root,p=xxx,h=mcamel-common-kpanda-m
ysql-cluster-mysql-0.mysql.mcamel-system,P=3306 dsn=u=root,p=xxx,h=mcamel-common-kpanda-m
ysql-cluster-mysql.mysql.mcamel-system,P=3306 --print
```

```
pt-table-sync --execute --charset=utf8 --ignore-databases=mysql,sys,percona --databases=kpanda
dsn=u=root,p=xxx,h=mcamel-common-kpanda-mysql-cluster-mysql-0.mysql.mcamel-system,P=3306
dsn=u=root,p=xxx,h=mcamel-common-kpanda-mysql-cluster-mysql-1.mysql.mcamel-system,P=3306
--print
```

使用 pt-table-sync 即可完成数据补充，示例中是 **mysql-0=> mysql-1** 补充数据。

这种场景往往适用于主从切换，发现新从库有多余的已执行的 gtid 在重做之前补充数据。

这种补充数据只能保证数据不丢失，如果新主库已经删除的数据会被重新补充回去，是一个潜在的风险，如果是新主库有数据，会被替换成老数据，也是一个风险。

其他故障

CR 创建数据库失败报错

数据库运行正常，使用 CR 创建数据库出现了报错，此类问题的原因有：`mysql root` 密码

有特殊字符

image
image

1. 获取查看原密码：

```
[root@master-01 ~]$ kubectl get secret -n mccamel-system mccamel-common-mysql-cluster-secret -o=jsonpath='{.data.ROOT_PASSWORD}' | base64 -d
```

2. 如果密码含有特殊字符 - ，进入 MySQL 的 Shell 输入原密码出现以下错误

```
bash-4.4# mysql -uroot -p
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
```

3. 清理重建：

- 方法一：清理数据目录，删除 Pod 等待 sidecar running 以后，再删除一次

数据目录，再删除 Pod 即可恢复：

```
[root@master-01 ~]# kubectl exec -it mccamel-common-mysql-cluster-mysql-1 -n
mccamel-system -c sidecar -- /bin/sh
sh-4.4# cd /var/lib/mysql
sh-4.4# ls | xargs rm -rf
```

- 方法二：先删除 PVC，再删除 Pod，即可恢复：

```
kubectl delete pvc data-mcamel-common-mysql-cluster-mysql-1 -n mccamel-system
kubectl delete pod mccamel-common-mysql-cluster-mysql-1 -n mccamel-system
```

!!! note

使用以上方法清理重建将导致数据库被重置，数据丢失。

提示 The MySQL server is running with the read-only option so it cannot execute this statement

当在管理平台的操作中产生如下提示，说明 MySQL 节点主从关系发生变化，但平台其他模块没有及时转换连接对象，在只读从节点执行了写操作。

The MySQL server is running with the read-only option so it cannot execute this statement
image

image

解决方法：前往 容器管理 平台重启所有相关 replica。

Operator 或者相关 MySQL 资源中出现错误码 1045

原因：磁盘性能太差，导致 MySQL 初始化被中断

当出现这个错误后，登录 MySQL，执行：

mysql -uroot

如果可以直接登录，很大概率是因为磁盘性能太差，导致 MySQL 初始化被中断。

临时解决方案

1. 将 mysql-operator 这个 StatefulSet 缩容为 0
2. 删除 MySQL 对应的 StatefulSet 里 MySQL 容器的 probe 探针
3. 删除 MySQL 的 PVC
4. 删除 MySQL 的 Pod，并等待 MySQL 重新初始化
5. (待 MySQL 启动成功后)，使用 mysql -uroot 登录 MySQL，查看是否可以登录。

如果无法登录，则说明 MySQL 初始化成功。

6. 将 mysql-operator 这个 StatefulSet 扩容为原来的值

MySQL MGR 参数配置

在配置 MySQL Group Replication (MGR) 时，为了增加配置的灵活性和向后兼容性，部分参数需要使用 `loose_` 前缀。MySQL 中有些参数可能是实验性的或仅用于特定场景。在这些情况下，使用 `loose_` 前缀可以确保这些参数在不支持的环境下被忽略。

具体示例

以下是一个使用 `loose_` 前缀配置 MySQL Group Replication 的示例：

[mysqld]

```
# Enable Group Replication
loose_group_replication_start_on_boot=off
loose_group_replication_bootstrap_group=off
loose_group_replication_group_name="aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee"
loose_group_replication_local_address="192.168.0.1: 33061"
loose_group_replication_group_seeds="192.168.0.1: 33061,192.168.0.2:33061,192.168.0.3:33061"
loose_group_replication_single_primary_mode=on
loose_group_replication_enforce_update_everywhere_checks=off

# Group Replication SSL settings (optional)
loose_group_replication_ssl_mode=REQUIRED
loose_group_replication_ssl_ca=ca.pem
loose_group_replication_ssl_cert=server-cert.pem
loose_group_replication_ssl_key=server-key.pem

# Other necessary settings
binlog_checksum=NONE
binlog_format=ROW
log_slave_updates=ON
gtid_mode=ON
enforce_gtid_consistency=ON
master_info_repository=TABLE
relay_log_info_repository=TABLE
transaction_write_set_extraction=XXHASH64
!!! Info
```

使用 `loose_` 前缀是配置 MGR 参数时的一种安全做法，尤其是在不同版本或插件状态下运行时。它可以确保配置文件的兼容性和灵活性，避免不必要的启动错误。

MySQL MGR 排障手册

常用命令

获取 root 密码

在 MySQL MGR 集群的命名空间下，查找以 -mgr-secret 结尾的 Secret 资源，这里以获取 kpanda-mgr 这个集群的 Secret 为例：

```
kubectl get secrets/kpanda-mgr-mgr-secret -n mcamel-system --template={{.data.rootPassword}} |  
base64 -d  
root123!
```

查看集群状态

通过 MySQL 命令行查看：

```
mysqlsh -uroot -pPassword -- cluster status  
{  
    "clusterName": "kpanda_mgr",  
    "defaultReplicaSet": {  
        "name": "default",  
        "primary": "kpanda-mgr-2.kpanda-mgr-instances.mcamel-system.svc.cluster.local:3306",  
        "ssl": "REQUIRED",  
        "status": "OK",  
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",  
        "topology": {  
            "kpanda-mgr-0.kpanda-mgr-instances.mcamel-system.svc.cluster.local:3306": {  
                "address": "kpanda-mgr-0.kpanda-mgr-instances.mcamel-system.svc.cluster.local:  
3306",  
                "memberRole": "SECONDARY",  
                "mode": "R/O",  
                "readReplicas": {},  
                "replicationLag": "applier_queue_applied",  
                "role": "HA",  
                "status": "ONLINE",  
                "version": "8.0.31"  
                "address": "kpanda-mgr-1.kpanda-mgr-instances.mcamel-system.svc.cluster.local:  
3306",  
                "memberRole": "PRIMARY",  
                "mode": "R/W",  
                "readReplicas": {},  
                "replicationLag": "0ms",  
                "role": "LEADER",  
                "status": "ONLINE",  
                "version": "8.0.31"  

```

```

    "address": "kpanda-mgr-1.kpanda-mgr-instances.mcamel-system.svc.cluster.local:3306",
        "memberRole": "SECONDARY",
        "mode": "R/O",
        "readReplicas": {},
        "replicationLag": "applier_queue_applied",
        "role": "HA",
        "status": "ONLINE",
        "version": "8.0.31"
    },
    "kpanda-mgr-2.kpanda-mgr-instances.mcamel-system.svc.cluster.local:3306": {
        "address": "kpanda-mgr-2.kpanda-mgr-instances.mcamel-system.svc.cluster.local:3306",
        "memberRole": "PRIMARY",
        "mode": "R/W",
        "readReplicas": {},
        "replicationLag": "applier_queue_applied",
        "role": "HA",
        "status": "ONLINE",
        "version": "8.0.31"
    }
},
"topologyMode": "Single-Primary"
},
"groupInformationSourceMember": "kpanda-mgr-2.kpanda-mgr-instances.mcamel-system.svc.cluster.local:3306"
}

```

!!! note

集群在正常情况下：

- 所有的节点的 status 都为 ONLINE 状态。
- 有一个节点的 memberRole 为 PRIMARY，其他节点都为 SECONDARY。

用 SQL 语句查看 : `SELECT * FROM performance_schema.replication_group_members\G`

```

mysql> SELECT * FROM performance_schema.replication_group_members\G
*****
 1. row ****
 CHANNEL_NAME: group_replication_applier
 MEMBER_ID: 6f464f4e-ba96-11ee-a028-a225dc125542
 MEMBER_HOST: kpanda-mgr-2.kpanda-mgr-instances.mcamel-system.svc.cluster.local
 MEMBER_PORT: 3306
 MEMBER_STATE: ONLINE
 MEMBER_ROLE: PRIMARY
 MEMBER_VERSION: 8.0.31

```

```

MEMBER_COMMUNICATION_STACK: MySQL
*****
2. row ****
    CHANNEL_NAME: group_replication_applier
        MEMBER_ID: b3a53102-bfec-11ee-a821-0a8fb9f7d1ce
        MEMBER_HOST: kpanda-mgr-0.kpanda-mgr-instances.mccamel-system.svc.cluster.local
    MEMBER_PORT: 3306
    MEMBER_STATE: ONLINE
    MEMBER_ROLE: SECONDARY
    MEMBER_VERSION: 8.0.31
MEMBER_COMMUNICATION_STACK: MySQL
*****
3. row ****
    CHANNEL_NAME: group_replication_applier
        MEMBER_ID: bddd16f-bfec-11ee-a7a4-324c8edaca40
        MEMBER_HOST: kpanda-mgr-1.kpanda-mgr-instances.mccamel-system.svc.cluster.local
    MEMBER_PORT: 3306
    MEMBER_STATE: ONLINE
    MEMBER_ROLE: SECONDARY
    MEMBER_VERSION: 8.0.31
MEMBER_COMMUNICATION_STACK: MySQL
3 rows in set (0.00 sec)

```

查看成员状态

```

查看成员状态 : SELECT * FROM performance_schema.replication_group_member_stats\G
mysql> SELECT * FROM performance_schema.replication_group_member_stats\G
*****
1. row ****
    CHANNEL_NAME: group_replication_applier
        VIEW_ID: 17066729271025607:9
        MEMBER_ID: 6f464f4e-ba96-11ee-a028-a225dc125542
        COUNT_TRANSACTIONS_IN_QUEUE: 0
        COUNT_TRANSACTIONS_CHECKED: 4748638
        COUNT_CONFLICTS_DETECTED: 0
        COUNT_TRANSACTIONS_ROWS_VALIDATING: 1109
        TRANSACTIONS_COMMITTED_ALL_MEMBERS: 6f464f4e-ba96-11ee-a028-a225dc125542:1-10,
95201c7d-ba96-11ee-a018-bed74fb0bf8d: 1-8,
b438e224-ba96-11ee-bc57-bed74fb0bf8d: 1-12516339,
b439a7d3-ba96-11ee-bc57-bed74fb0bf8d: 1-18
        LAST_CONFLICT_FREE_TRANSACTION: b438e224-ba96-11ee-bc57-bed74fb0bf8d:12519298
        COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE: 0

```

```

COUNT_TRANSACTIONS_REMOTE_APPLIED: 6
COUNT_TRANSACTIONS_LOCAL_PROPOSED: 4748638
COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 0
*****
2. row *****
    CHANNEL_NAME: group_replication_applier
    VIEW_ID: 17066729271025607:9
    MEMBER_ID: b3a53102-bfec-11ee-a821-0a8fb9f7d1ce
    COUNT_TRANSACTIONS_IN_QUEUE: 0
    COUNT_TRANSACTIONS_CHECKED: 4514132
    COUNT_CONFLICTS_DETECTED: 0
    COUNT_TRANSACTIONS_ROWS_VALIDATING: 1110
    TRANSACTIONS_COMMITTED_ALL_MEMBERS: 6f464f4e-ba96-11ee-a028-a225dc12
5542:1-10,
95201c7d-ba96-11ee-a018-bed74fb0bf8d: 1-8,
b438e224-ba96-11ee-bc57-bed74fb0bf8d: 1-12519027,
b439a7d3-ba96-11ee-bc57-bed74fb0bf8d: 1-18
        LAST_CONFLICT_FREE_TRANSACTION: b438e224-ba96-11ee-bc57-bed74fb0bf8
d:12520590
    COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE: 0
    COUNT_TRANSACTIONS_REMOTE_APPLIED: 4514129
    COUNT_TRANSACTIONS_LOCAL_PROPOSED: 0
    COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 0
*****
3. row *****
    CHANNEL_NAME: group_replication_applier
    VIEW_ID: 17066729271025607:9
    MEMBER_ID: bddd16f-bfec-11ee-a7a4-324c8edaca40
    COUNT_TRANSACTIONS_IN_QUEUE: 0
    COUNT_TRANSACTIONS_CHECKED: 4658713
    COUNT_CONFLICTS_DETECTED: 0
    COUNT_TRANSACTIONS_ROWS_VALIDATING: 1093
    TRANSACTIONS_COMMITTED_ALL_MEMBERS: 6f464f4e-ba96-11ee-a028-a225dc12
5542:1-10,
95201c7d-ba96-11ee-a018-bed74fb0bf8d: 1-8,
b438e224-ba96-11ee-bc57-bed74fb0bf8d: 1-12519027,
b439a7d3-ba96-11ee-bc57-bed74fb0bf8d: 1-18
        LAST_CONFLICT_FREE_TRANSACTION: b438e224-ba96-11ee-bc57-bed74fb0bf8
d:12520335
    COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE: 0
    COUNT_TRANSACTIONS_REMOTE_APPLIED: 4658715
    COUNT_TRANSACTIONS_LOCAL_PROPOSED: 0
    COUNT_TRANSACTIONS_LOCAL_ROLLBACK: 0
3 rows in set (0.00 sec)

```

指定成员角色

1. 将某个节点指定为 PRIMARY。

```
select group_replication_set_as_primary('4697c302-3e52-11ed-8e61-0050568a658a');
```

2. mysqlsh 语法

```
JS > var c=dba.getCluster()
JS > c.status()
JS > c.setPrimaryInstance('172.30.71.128:3306')
```

常见故障

某个 SECONDARY 节点为非 ONLINE 状态

```
{
  "clusterName": "mgr0117",
  "defaultReplicaSet": {
    "name": "default",
    "primary": "mgr0117-2.mgr0117-instances.m0103.svc.cluster.local:3306",
    "ssl": "REQUIRED",
    "status": "OK_NO_TOLERANCE_PARTIAL",
    "statusText": "Cluster is NOT tolerant to any failures. 1 member is not active.",
    "topology": {
      "mgr0117-0.mgr0117-instances.m0103.svc.cluster.local:3306": {
        "address": "mgr0117-0.mgr0117-instances.m0103.svc.cluster.local:3306",
        "instanceErrors": [
          "NOTE: group_replication is stopped."
        ],
        "memberRole": "SECONDARY",
        "memberState": "OFFLINE",
        "mode": "R/O",
        "readReplicas": {},
        "role": "HA",
        "status": "(MISSING)",
        "version": "8.0.31"
      },
      "mgr0117-1.mgr0117-instances.m0103.svc.cluster.local:3306": {
        "address": "mgr0117-1.mgr0117-instances.m0103.svc.cluster.local:3306",
        "memberRole": "SECONDARY",
        "mode": "R/O",
        "readReplicas": {}
      }
    }
  }
}
```

```

    "replicationLag": "applier_queue_applied",
    "role": "HA",
    "status": "ONLINE",
    "version": "8.0.31"
  },
  "mgr0117-2.mgr0117-instances.m0103.svc.cluster.local:3306": {
    "address": "mgr0117-2.mgr0117-instances.m0103.svc.cluster.local:3306",
    "memberRole": "PRIMARY",
    "mode": "R/W",
    "readReplicas": {},
    "replicationLag": "applier_queue_applied",
    "role": "HA",
    "status": "ONLINE",
    "version": "8.0.31"
  }
},
"topologyMode": "Single-Primary"
},
"groupInformationSourceMember": "mgr0117-2.mgr0117-instances.m0103.svc.cluster.local:330
6"
}

```

这里看到对应的 address 字段是 mgr0117-0.mgr0117-instances.m0103.svc.cluster.local: 3306 ,

进入 mgr0117-0 这个 pod , 执行

```
mysql> start group_replication;
Query OK, 0 rows affected (5.82 sec)
```

这里如果数据量比较大 , 该节点会处于比较长时间的 RECOVERING 状态。

没有 PRIMARY 节点 , 各个节点都显示 OFFLINE

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+-----+
| CHANNEL_NAME          | MEMBER_ID | MEMBER_HOST | MEMBER_PORT |
-----+-----+-----+-----+-----+-----+
| MEMBER_STATE | MEMBER_ROLE | MEMBER_VERSION | MEMBER_COMMUNICATIO
N_STACK |
+-----+-----+-----+-----+-----+-----+
| group_replication_applier |           |           |      NULL | OFFLINE   |
|                           |           |           |           |           |
+-----+-----+-----+-----+-----+-----+
```

```
-----+  
1 row in set (0.00 sec)
```

此时可尝试从 mysql shell 重启集群：

```
dba.rebootClusterFromCompleteOutage().
```

若依然不能解决，则使用 cmd 方式登录之前的 PRIMARY 的节点，然后启动该节点的 group replication：

```
set global group_replication_bootstrap_group=on;  
start group_replication;  
set global group_replication_bootstrap_group=off;  
!!! warning
```

对于其他节点，依次执行上面的命令。

MGR 的 Pod 一直处于 terminating 状态

需要检查：

1. Kubernetes 集群的各个组件可能不正常，检查相关组件状态，尤其是 etcd
2. MGR 的 Operator 状态是否正常，检查 Operator Pod 的日志

如果处于测试环境，需要快速删除 Pod，可以删除相关 pod 的 finalizers：

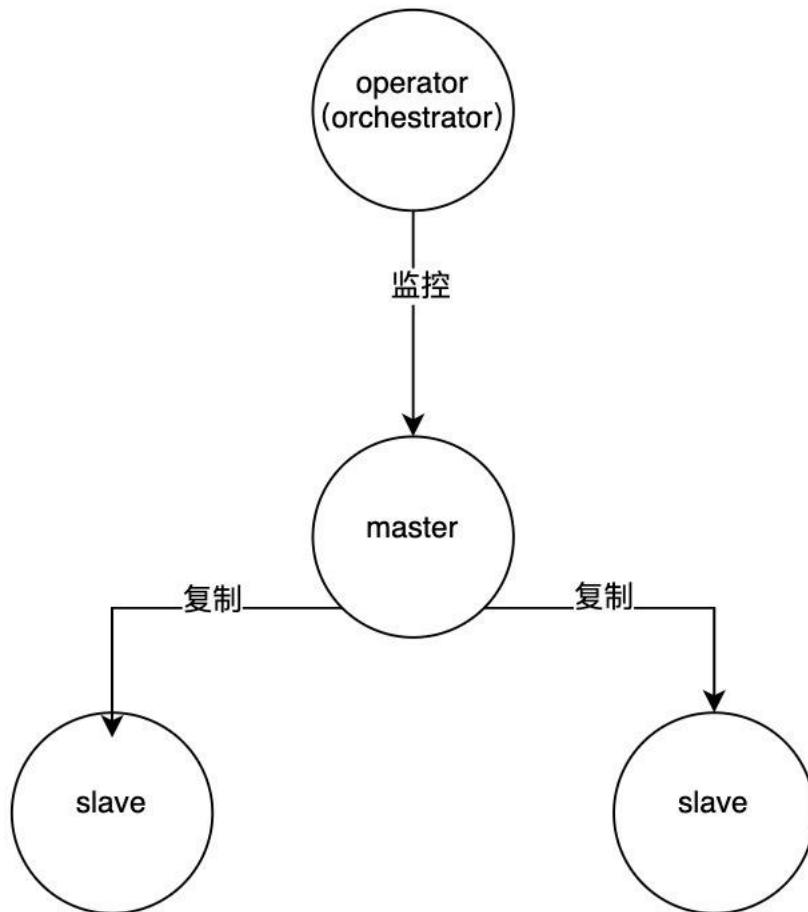
```
kubectl edit pod <your-pod-name>
```

在 Pod 的 YAML 中删除以下几行：

```
finalizers:  
- mysql.oracle.com/membership  
- kopf.zalando.org/KopfFinalizerMarker
```

MySQL 主从模式应对网络闪断

MySQL 主从模式的高可用保障是独立于集群的，这就可能存在误判，例如下面的集群：



00

假设 master 的网络发生短暂闪断（时间大于 orch 切换的容忍时间），orch 会做 failover，将其中一个 slave 提升为 master。然而 master 的网络有可能在切换后不久就恢复了，这时这个切换操作就是多余的。

为了应对这种场景，我们可以停止 orch 对该 MySQL 集群的自动切换能力。

!!! note

适用于集群网络状态不可控的情况。原理是在 orch 监控到 master 网络不可达后，将其忽略。

操作步骤

1. 以 common-mysql 数据库，使用 helm 更新 operator：

```
helm -n mcamel-system get values mysql-operator > values.yaml
```

2. 获取之前安装版本设置的 value，再升级 mysql-operator。

```
helm upgrade \
--install mysql-operator \
```

```
--create-namespace \
-n mcamel-system \
--cleanup-on-fail mcamel-release/mysql-operator \
--version 0.14.0-rc2 \
-f values.yaml \
--set "orchestrator.config.RecoveryIgnoreHostnameFilters[0]=^mcamel-common" # (1)!
```

1. 这里是一个正则，则最终会和 mysql pod 名字做匹配

!!! note

确保执行完成后，operator 发生了重启；

确保执行完成后，operator 的配置文件，名为 mysql-operator-orc 的 configmap 中有 --set 的内容。

验证方案

1. 建立一个 3 节点主从集群（-0 是 master），名字匹配上面设置的正则：

mcamel-common-test-cluster-mysql-0	●	4/4	Running	2	192.168.180.24	master2	7m24s
mcamel-common-test-cluster-mysql-1	●	4/4	Running	1	192.168.84.221	master0	6m46s
mcamel-common-test-cluster-mysql-2	●	4/4	Running	0	192.168.137.106	master1	5m58s

01

2. 停止 master 的网络；

```

NAME= ens3
PF    READY STATUS   RESTARTS IP           NODE   AGE
mcamel-common-test-cluster-mysql-0  brd  3/4   Running   3  192.168.180.24  master2  19m
mcamel-common-test-cluster-mysql-1  brd  4/4   Running   1  192.168.84.221 master0  19m
mcamel-common-test-cluster-mysql-2  brd  4/4   Running   0  192.168.137.106 master1  18m

pod: ~

root@master2:~ (ssh)
valid_lft forever preferred_lft forever
3120: brd:1f1c93d15bf4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc mq state UP group default
link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 8
inet6 fe80::ecce:eff:feee:eee/64 scope link
valid_lft forever preferred_lft forever
3128: brd:1f28a40700f4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc mq state UP group default
link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 3
inet6 fe80::ecce:eff:feee:eee/64 scope link
valid_lft forever preferred_lft forever
3019: brd:1f28a40700f4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc mq state UP group default
link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 2
inet6 fe80::ecce:eff:feee:eee/64 scope link
valid_lft forever preferred_lft forever
3025: brd:1f474d764015bf4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc mq state UP group default
link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 1
inet6 fe80::ecce:eff:feee:eee/64 scope link
valid_lft forever preferred_lft forever
[root@master2 ~]#
[root@master2 ~]#
[root@master2 ~]# sudo ip link set brd:1f28a40700f4 down

```

01

3. 可以看到 master 没有发生切换；

mcamel-common-test-cluster-mysql-0	2/4	CrossLoopBackOff	9 (14s ago)	32m	192.168.180.24	master2	<none>	<none>	yes master
mcamel-common-test-cluster-mysql-1	4/4	Running	0	31m	192.168.84.221	master0	<none>	<none>	no replica
mcamel-common-test-cluster-mysql-2	4/4	Running	0	30m	192.168.137.106	master1	<none>	<none>	no replica

01

4. 可以看到集群已经恢复正常，master 仍然是 -0。

```

* - sshpass -p '123456' ssh -o StrictHostKeyChecking=no root@10.6.231.12
Last login: Tue Apr 16 22:57:09 2024 from 10.70.4.4
[root@master2 ~]# sudo ip link set cali735a3fd0223 up
[root@master2 ~]#
[×] root@master0: (ssh)
[root@master0 ~]# kubectl get po -o wide -lhealthy,role | grep mcomel-common-test-cluster
mcomel-common-test-cluster-mysql-0           4/4   Running   10 (5m53s ago)   37m   192.168.180.24   master2   <none>      <none>      yes   master
mcomel-common-test-cluster-mysql-1           4/4   Running   0          37m   192.168.84.221   master0   <none>      <none>      yes   replica
mcomel-common-test-cluster-mysql-2           4/4   Running   0          36m   192.168.137.106   master1   <none>      <none>      yes   replica

```

01

!!! note

另外再验证，名字不匹配上面正则的集群，同样的步骤，可以正常切换。

什么是 MongoDB

MongoDB 是一种面向文档的 NoSQL 数据库管理系统，它以灵活的数据模型和可扩展性而闻名。下面是 MongoDB 的一些重要特点和概念：

面向文档的数据模型：MongoDB 存储数据的方式是以文档的形式，文档是一种类似于 JSON 的结构，它是一个键值对的集合，可以嵌套和包含多种类型的数据。这种灵活的数据模型使得 MongoDB 非常适合存储复杂的、非结构化的数据。

分布式架构和高可用性：MongoDB 能够水平扩展到多个服务器，以应对大规模数据和高并发请求。它支持主从复制和分片（sharding）机制，可以实现数据的冗余备份和负载均衡，提供高可用性和扩展性。

查询和索引：MongoDB 提供丰富的查询功能，支持多种查询条件和操作符。它还支持灵活的索引机制，可以根据需要创建多个索引来优化查询性能。

强大的数据处理功能：MongoDB 内置了许多数据处理功能，例如聚合管道（aggregation pipeline）、地理空间索引和查询、全文搜索等。这些功能使得 MongoDB 可以在数据库层面上处理复杂的数据操作和分析任务。

开发者友好性：MongoDB 提供了丰富的 API 和驱动程序，支持多种编程语言。它还具有简单而直观的命令行界面和图形用户界面，使开发人员能够快速上手和使用。

MongoDB 被广泛应用于 Web 应用、大数据分析、物联网、实时数据处理等领域，它的灵

活性和可扩展性使得开发人员能够更高效地处理和存储不断增长的数据。

创建

创建

功能特性

MongoDB 拥有丰富的功能，以下是一些它的主要功能：

- 高可用性和冗余备份

MongoDB 支持主从复制和副本集（replica set）机制，保证数据的冗余备份和高可用性。主从复制通过将数据从主节点复制到从节点，实现数据的持久性和故障恢复；副本集是一组互相通信的 MongoDB 实例，当主节点不可用时，自动选举新的主节点。

- 水平扩展和负载均衡

MongoDB 支持分片（sharding）机制，可以将数据水平分割并分布在多个服务器上。这样可以实现数据的横向扩展，提高系统的存储容量和处理能力。MongoDB 提供自动的数据分布和负载均衡，使得数据的访问和查询在整个集群中得到均衡分配。

- 强大的查询和索引功能

MongoDB 支持丰富的查询条件和操作符，可以进行灵活的查询。它还支持多种索引类型，包括单键索引、复合索引、文本索引和地理空间索引等，以提高查询性能和响应速度。

- 数据处理和聚合管道

MongoDB 提供了强大的数据处理功能，通过聚合管道（aggregation pipeline）可以进行复杂的数据操作和分析。聚合管道允许串联多个操作，如过滤、排序、分组、计

算等，以获得所需的结果。

- 地理空间和全文搜索

MongoDB 支持地理空间数据的存储和查询，可以进行地理位置相关的操作和搜索。

同时，它还提供了全文搜索的功能，可以进行文本关键字的搜索和索引。

- 完整的事务支持

MongoDB 从版本 4.0 开始引入了多文档事务的支持。这允许开发人员在一个或多个

文档上执行原子性的读写操作，并保持数据的一致性。

- 安全性和权限管理

MongoDB 提供了丰富的安全性功能，支持身份验证、访问控制和加密通信等。开发者可以为用户和角色定义不同的权限级别，以保护数据的安全性。

以上是 MongoDB 的一些主要功能，它的灵活性、可扩展性和丰富的功能使得开发人员能够应对各种不同的数据管理和处理需求。

离线升级中间件 - Mongodb 模块

本页说明从[下载中心](#)下载中间件 - Mongodb 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `_mcamel_` 字样是中间件模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 `chart-syncer` 同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

==== “已安装 chart repo”

若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
 url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```

```

==== “未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: LOCAL
```

```

```

path: ./local-repo # chart 本地路径
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

## 2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

### 1. 解压 tar 压缩包。

```

tar -xvf mccamel-mongodb_0.3.1_amd64.tar
cd mccamel-mongodb_0.3.1_amd64
tar -xvf mccamel-mongodb_0.3.1.bundle.tar

```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

### 2. 从本地加载镜像到 Docker 或 containerd。

```

==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

### !!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

==== “通过 helm repo 升级”

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep mongodb
```
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-mongodb http://{harbor url}/chartrepo/{project}
```
```

```

1. 更新 helm 仓库。

```
```shell
helm repo update mcamel/mcamel-mongodb # helm 版本过低会导致失败，若失败，请尝试执行 helm update repo
```
```

```

1. 选择您想安装的版本（建议安装最新版本）。

```
```shell
helm search repo mcamel/mcamel-mongodb --versions
```
```
```none
[root@master ~]# helm search repo mcamel/mcamel-mongodb --versions
NAME CHART VERSION APP VERSION DESCRIPTION
mcamel/mcamel-mongodb 0.3.1 0.3.1 A Helm chart for Kubernetes
...
```
```

```

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-mongodb -n mccamel-system -o yaml > mccamel-mongodb.yaml
```

```

1. 执行 `helm upgrade` 。

升级前建议您覆盖 `mcamel-mongodb.yaml` 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-mongodb mcamel/mcamel-mongodb \
-n mccamel-system \
-f ./mcamel-mongodb.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.3.1
```

```

#### ==== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-mongodb -n mccamel-system -o yaml > mccamel-mongodb.yaml
```

```

1. 执行 `helm upgrade` 。

升级前建议您覆盖 `bak.yaml` 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```

```
```shell
helm upgrade mcamel-mongodb . \
-n mccamel-system \
-f ./mcamel-mongodb.yaml \

```

```
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```

```

# MongoDB Release Notes

本页列出 MongoDB 数据库的 Release Notes，便于您了解各版本的演进路径和特性变化。

\*[Mcamel-MongoDB]: Mcamel 是 DaoCloud 所有中间件的开发代号，MongoDB 是面向文档的 NoSQL 数据库中间件

## 2024-09-30

### v0.14.0

- **优化** 备份恢复的新实例中 Express 节点的服务类型默认为 NodePort
- **修复** 选择工作空间查询 MongoDB 列表时权限泄漏的问题
- **修复** 部分操作无审计日志的问题
- **修复** 恢复 MongoDB 失败的问题

## 2024-08-31

### v0.13.0

!!! warning

升级 mongodb-operator 到 v0.10.0 会导致已有的实例进行重启。

- **优化** 创建实例时不可选择异常的集群
- **优化** 恢复的新实例的控制台访问方式默认为 Nodeport
- **优化** 升级 mongodb-operator 版本到 0.10.0，基础镜像均使用 ubi-minimal: 8.6-994。

- **修复** 恢复 MongoDB 实例失败的问题

**2024-07-31**

**v0.12.0**

- **新增** 查询 MySQL 慢日志功能

**2024-05-31**

**v0.10.0**

- **新增** 参数模板导入功能
- **优化** 支持批量修改实例参数
- **优化** 删除备份时可选是否删除 S3 中备份数据

**2024-04-30**

**v0.9.0**

- **优化** 增加命名空间配额的提示
- **优化** 删除备份的时候，可以选择删除在 s3 里的备份数据
- **修复** 某些场景下 mongodb-agent readinessProbe 超时的问题
- **修复** 实例的配置参数搜索返回结果不正确

**2024-03-31****v0.8.0**

- **新增** 支持 MongoDB 实例的备份和恢复
- **新增** 支持参数模板
- **新增** 支持通过模板创建 MongoDB 实例
- **修复** mongodb 备份源集群不存在的时候的问题

**2024-01-31****v0.7.0**

- **优化** 在全局管理中增加 MongoDB 版本展示

**2023-12-31****v0.6.0**

- **新增** 监控面板支持中文
- **修复** 创建实例时部分输入框填写特殊字符的校验未生效的问题

**2023-11-30****v0.5.0**

- **新增** Mcamel-MongoDB 开启控制台 MongoDB Express

**2023-10-31****v0.4.0**

- **新增** 离线升级。
- **新增** 实例重启功能。
- **修复** cloudshell 权限问题。

**2023-08-31****v0.3.0**

- **优化** KindBase 语法兼容。
- **优化** operator 创建过程的页面展示。

**2023-07-31****v0.2.0**

- **新增** 对 log 目录所在 pvc 的配置能力。
- **优化** 创建实例对话框增加反亲和默认标签，简化配置过程。
- **优化** 可以在非 Operator 所在命名空间创建 MongoDB 实例。
- **优化** 增加前端界面权限相关的展示信息

**2023-06-30****v0.1.0**

- 新增 Mcamel-MongoDB 支持 MongoDB 的实例创建、查看、删除等管理功能

## 创建 MongoDB 实例

接入 MongoDB 数据库后，参照以下步骤创建 MongoDB 实例。

1. 在实例列表中，点击右上角的 **新建实例**。

创建

创建

2. 在 **创建 MongoDB 实例** 页面中，配置 **基本信息** 后，点击 **下一步**。

创建

创建

3. 选择部署类型、CPU、内存和存储等 **规格配置** 后，点击 **下一步**。

创建

创建

4. 设置用户名和密码等 **服务设置**，默认采用 ClusterIP 作为访问方式，用户还可以配

置 **亲和性**、**配置参数** 等多项设置，可在 **通用设置** 中查看。

创建

创建

创建

创建

5. 确认基本信息、规格配置、服务设置的信息准确无误后，点击 **确定**。

创建

创建

6. 返回实例列表，屏幕将提示 **创建实例成功**。

查看

查看

## 更新 MongoDB 实例

如果想要更新或修改 MongoDB 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择 **更新实例**。

更新实例

更新实例

2. 基本信息：只能修改描述。然后点击 **下一步**。

基本信息

基本信息

3. 修改规格配置后点击 **下一步**。

规格配置

规格配置

4. 修改服务设置后点击 **确定**。

服务设置

服务设置

# 删除 MongoDB 实例

如果想要删除一个 MongoDB 实例，可以执行如下操作：

1. 在 MongoDB 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择 **删除实例**。

删除实例

删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击 **删除** 按钮。

删除实例

删除实例

**!!! warning**

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

# 查看实例

在 MongoDB 实例列表中，选择想要查看的实例，点击实例名称进入详情页面。

## 概览

概览页面支持查看基本信息、访问设置、资源配额、主从同步延迟、监控告警、容器组列

表、最近事件等。其中，

- **主从同步延迟**：是指主从复制或同步过程中由于网络延迟或其他因素导致的从节点接收到主节点发送的数据包的时间差。
- **监控告警**：在概览页面仅支持查看最近 10 条告警，点击 **查看更多** 可跳转至告警列表。

## 概览

## 配置参数

在左侧导航栏点击 **配置参数**，点击页面右上角 **更新** 即可更新 当前值。

| 参数名称                                     | 默认值              | 当前值              | 可选值                       | 是否需要重启 | 描述                           |
|------------------------------------------|------------------|------------------|---------------------------|--------|------------------------------|
| net.compression.compressors              | snappy,zstd,zlib | snappy,zstd,zlib | {snappy/zstd/zlib}        | 是      | 设置mongod或mongos的网络压缩算法。      |
| net.serviceExecutor                      | synchronous      | synchronous      | {synchronous/adaptive}    | 是      | 决定mongod或mongos使用的线程和执行...   |
| net.tls.mode                             | disabled         | disabled         | {disabled/allowTLS/pre... | 是      | 启用用于所有网络连接的TLS。              |
| operationProfiling.mode                  | off              | off              | {off/slowOp/all}          | 是      | 指定应分析哪些操作。                   |
| operationProfiling.slowOpThresholdMs     | 100              | 100              | {0-}                      | 是      | 慢操作时间阈值，运行时间超过此阈值的...        |
| replication.enableMajorityReadConcern    | true             | true             | {true/false}              | 是      | 是否开启majority read concern支持。 |
| replication.oplogSizeMB                  | 2048             | 2048             | {2048-}                   | 是      | 复制操作日志的最大大小（以兆字节为单...        |
| setParameter.cursorTimeoutMillis         | 600000           | 600000           | {1-2147483647}            | 是      | 空闲游标的到期阈值，单位为毫秒。如果...        |
| setParameter.flowControlTargetLagSeco... | 10               | 10               | {1-600}                   | 是      | 流控下最多数据提交延迟目标值。              |
| setParameter.internalQueryExecMaxBloc... | 33554432         | 33554432         | {33554432-2684354...      | 是      | 排序阶段可能使用的最大内存，单位为字...        |
| setParameter.maxTransactionLockReque...  | 5                | 5                | {0-60}                    | 是      | 事务加锁超时时间。                    |

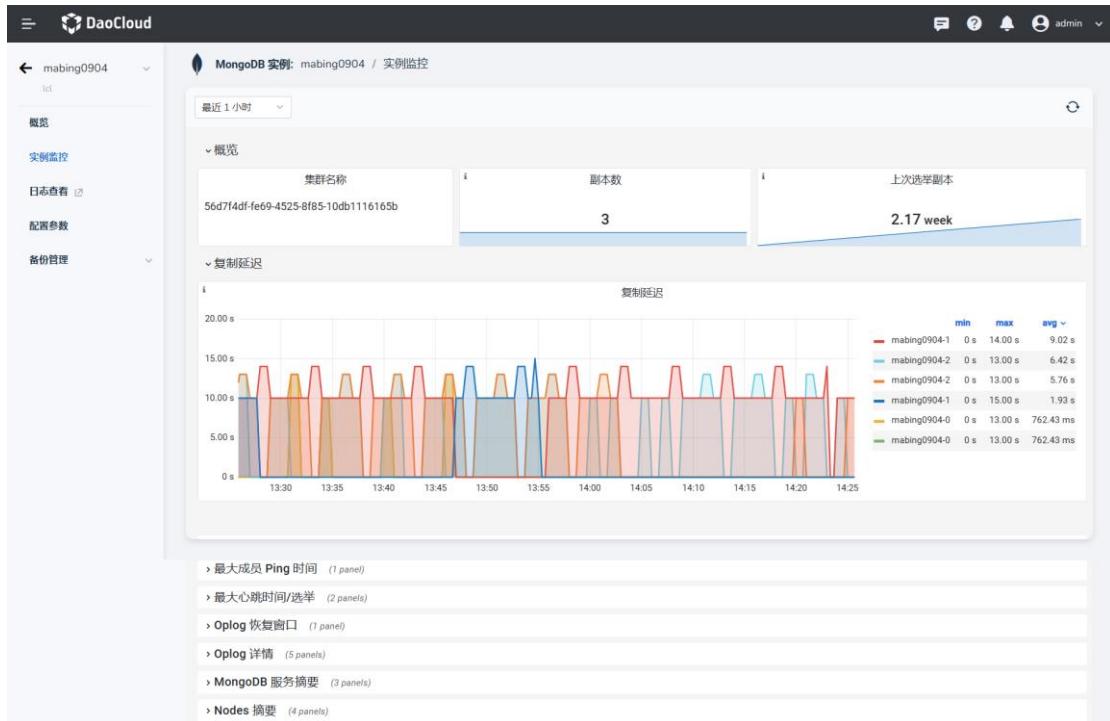
## 配置参数

### !!! warning

参数更新需要重启实例，重启会导致实例不可用，请谨慎操作！

## 实例监控

在左侧导航栏，点击 实例监控，可以接入监控模块。



实例监控

## 查看 MongoDB 日志

### 操作步骤

通过访问每个 MongoDB 的实例详情页面，可以支持查看 MongoDB 的日志。

1. 在 MongoDB 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。

日志

日志

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。

## 日志

### 日志

3. 点击 **日志查看** 即可进入到日志查看页面（[Insight 日志查看](#)）。

## 日志查看说明

在日志查看页面可以很方便的进行日志查看。常用操作说明如下：

- 支持自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持查看日志的上下文，点击右侧 **上下文** 图标即可
- 支持导出日志

## 备份管理

MongoDB 支持对运行中的实例数据进行自动或手动备份。

## 备份配置

您可以针对不同的实例创建多个备份配置。例如一个实例一个备份配置。

1. 在 MongoDB 数据库首页，点击 **配置管理**

The screenshot shows the MongoDB database management interface. It lists two instances: 'mongo327-test' and 'new-mo327'. Both instances are in the 'mcamel-cluster205' namespace and are currently running. The 'mongo327-test' instance has a '副本集' (Replica Set) type, 3 replicas, and resource allocations of 1 Core CPU, 0.5 GB memory, and 1 GB disk storage. The 'new-mo327' instance also has a '副本集' type, 3 replicas, and similar resource allocations. The interface includes a search bar, configuration management buttons, and a navigation bar with the DaoCloud logo.

## 配置管理

2. 在 **存储配置** 页签中点击 **创建** 按钮。

The screenshot shows the 'Create Configuration' page. It includes fields for configuration name ('名称' - 'bake-config'), backup type ('备份类型' - '托管 MinIO'), and backup instances ('备份实例'). It also includes fields for address ('地址' - 'http://10.233.7.122:32529'), Access Key ('Access\_Key' - 'geroi982393'), Secret Key ('Secret\_Key' - '982393'), and Bucket name ('Bucket 名称' - 'test'). A '校验' (Check) button is located next to the Bucket name field. At the bottom right, there are '取消' (Cancel) and '确定' (Confirm) buttons.

## 备份配置

- 备份类型：托管 MinIO 或 S3
- 备份实例：选择一个实例
- 输入访问实例的 Access\_Key、Secret\_Key 和 Bucket 名称

### !!! note

在 \*\*参数模板\*\* 页签中，可以创建各种模板用于后续配置。

3. 确认无误后点击 **确定**。

## 备份设置

备份之前，先要做好备份设置。

进入某个 MongoDB 实例，从左侧导航栏点击 备份管理 -> 备份设置。

### 备份设置

- 备份配置：可以预先创建多个[备份配置](#)
- 路径：要备份数据的路径，例如 /data123
- 自动备份：开启自动备份后，将自动全量备份运行中的实例

自动备份默认可保留 30 份。

## 创建备份

1. 进入某个 MongoDB 实例，从左侧导航栏点击 备份管理 -> 备份数据 -> 创建备份

按钮

The screenshot shows the 'MongoDB 实例: mabing0325 / 备份数据' (MongoDB Instance: mabing0325 / Backup Data) page. A modal window titled '创建备份' (Create Backup) is open, displaying the message '备份名称由用户自定义及创建备份数据时的时间戳组成。' (Backup name is user-defined and includes a timestamp at the time of backup creation). Below it, a table lists one backup entry:

| 备份名称                | 备份状态 | 备份方式 | 备份开始时间-备份结束时间                       | 备份路径                         | 备份类型 |
|---------------------|------|------|-------------------------------------|------------------------------|------|
| agent123-1711530287 | 备份成功 | 手动   | 2024-03-27 17:04 - 2024-03-27 17:04 | s3://mabing/database/aaa1... | 全量备份 |

## 创建备份

2. 在弹窗中输入一个备份名称后点击 **确定**

The screenshot shows the '创建备份' (Create Backup) modal window. It contains a text input field labeled '备份名称' (Backup Name) with the value 'bake0328'. Below the input field is a note: '4~64个字符之间，必须以字母开头，区分大小写，可以包含字母、数字、中划线或者下划线，不能包含其他特殊字符。' (4~64 characters, must start with a letter, case-sensitive, can contain letters, numbers, underscores, or hyphens, cannot contain other special characters.). At the bottom right of the modal are two buttons: '取消' (Cancel) and '确定' (Confirm).

输入名称

3. 屏幕显示创建成功提示，点击右侧的 **更多操作** 按钮，可以执行更多操作。

## 恢复数据

如果想要恢复数据，可以在备份数据列表中，点击右侧的 **更多操作** 按钮，在弹出菜单中选择 **恢复**。

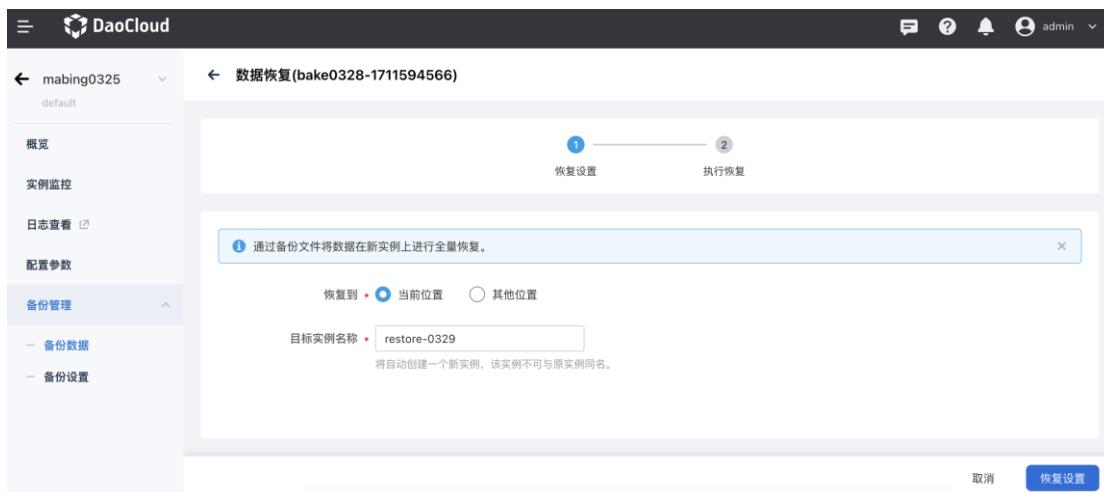
The screenshot shows the 'MongoDB 实例: mabing0325 / 备份数据' (MongoDB Instance: mabing0325 / Backup Data) page. The backup list table includes the following entries:

| 备份名称                | 备份状态 | 备份方式 | 备份开始时间-备份结束时间                       | 备份路径                         | 备份类型 |
|---------------------|------|------|-------------------------------------|------------------------------|------|
| agent123-1711530287 | 备份成功 | 手动   | 2024-03-27 17:04 - 2024-03-27 17:04 | s3://mabing/database/aaa1... | 全量备份 |
| bake0328-1711594566 | 备份成功 | 手动   | 2024-03-28 10:55 - 2024-03-28 10:55 | s3://mabing/database/aaa1... | 全量备份 |

A context menu is open over the second backup entry ('bake0328-1711594566'), with the '恢复' (Restore) option highlighted.

恢复

选择恢复的位置，输入目标实例的名称后，点击 **恢复设置**。



**恢复**

## 什么是 PostgreSQL

[PostgreSQL](#) 是一种开源的关系型数据库管理系统（RDBMS），它采用了美国加州大学伯克利分校开发的 Postgres 作为基础，并由 PostgreSQL 全球开发团队进行开发和维护。它是一款功能丰富、可扩展性强、安全可靠、兼容性好的数据库，已广泛应用于企业、互联网公司和开源社区中。

PostgreSQL 的特点包括：

- 具备完整的 ACID 事务支持，支持多版本并发控制（MVCC）机制，保证数据一致性。
- 能够灵活地支持各种数据类型，包括数值、日期、时间、JSON、XML 等。
- 提供灵活的扩展机制，包括自定义函数、PL/SQL 存储过程、外部表等。
- 能够支持复杂查询，包括聚合、子查询、连接等。
- 支持存储过程和触发器，以实现数据库应用逻辑的封装。
- 具备高度安全性，包括 SSL 加密、数据的备份和自动恢复机制等。
- 提供跨平台的支持，可以在各种操作系统下运行，包括 Linux、Windows、Unix 等。

- 具备高可用性和可扩展性，支持主从复制、流复制和连接池等机制。

PostgreSQL 还有一些其他的特点，如稳定性强、性能优异、易于部署和管理等。它也被广泛应用于大规模的企业级应用开发、互联网应用和数据仓库等领域。

在 PostgreSQL 社区中，有大量的贡献者和开发者，他们为 PostgreSQL 项目做出了各种各样的贡献，包括功能开发、错误修复、文档编写等。PostgreSQL 社区也提供了各种资源和支持，包括邮件列表、论坛、官方文档等，对开发者提供全面的帮助和支持。

### 容器化 PostgreSQL

容器化技术通过将应用和其依赖项打包在一个独立的运行环境中，从而实现了一种轻量级、便携和可重复性的部署方式。这种技术对于 PostgreSQL 数据库的安装、配置和管理非常有用。

- 简化部署和管理：容器化技术可以轻松地对 PostgreSQL 进行安装、配置和部署，减少开发人员和系统管理员的工作量。
- 快速复制和扩展：容器化 PostgreSQL 能够很容易地进行复制和扩展，从而满足大规模数据集的需求。
- 跨平台支持：由于容器化是一种轻量级的虚拟化技术，因此可以在任何支持该技术的服务器上运行。
- 可靠性和安全性：容器化技术提供了一个隔离的运行环境，可以有效地保护 PostgreSQL 实例免受外部攻击和运行环境的影响。

### [创建 PostgreSQL 实例](#)

## 技术特性及基本概念

DCE 5.0 商业版中提供的容器化 PostgreSQL 数据库具有以下特性。

- **多版本并发控制 ( MVCC )**

PostgreSQL 的 MVCC 机制使得并发控制能力得到了很大的提升，允许事务以一种非阻塞的方式执行，从而避免了死锁的情况。

- **复制和高可用性**

支持多种复制和高可用性方案，包括主从复制、流复制和逻辑复制等。这些方案可以提供数据冗余和自动故障转移等能力，从而提高系统的可用性。

- **安全性**

提供严格的安全保护措施，包括访问控制列表 ( ACL )、加密传输、密码认证和审计日志等。这些措施能够保证数据库数据的安全和完整性。

- **扩展性**

允许用户通过扩展来增强其功能。例如，用户可以自定义类型、自定义函数、自定义操作符和自定义索引等等。这些扩展能够增强 PostgreSQL 的能力和适应性。

- **性能优化**

良好的性能优化机制，可以通过各种手段对性能进行优化，包括索引优化、查询优化和配置优化等。同时，PostgreSQL 提供了强大的统计信息和实时监控工具，可以帮助用户进行性能问题的排查和优化。

- **支持 JSON 和 JSONB 数据类型**

支持直接存储和处理 JSON 格式的数据，这让 PostgreSQL 成为一个非常有用的 NoSQL 数据库。

- **支持全文检索**

内置全文检索功能，可以对大量的文本数据进行高效的搜索和匹配。

- **支持地理信息数据类型**

支持对地理信息数据进行存储和查询，这让 PostgreSQL 成为一个非常有用的 GIS 数据库。

- **支持分区表**

支持将一个大型的表分成多个小型表进行存储和查询，这可以提高查询效率和管理方便性。

- **支持并发控制**

通过 MVCC 机制来提高并发控制能力，同时 PostgreSQL 还支持多种隔离级别，包括 Read Committed、Repeatable Read 和 Serializable 等。

- **支持 PL/SQL 和 PL/Python 等存储过程语言**

支持多种存储过程语言，包括 PL/SQL、PL/Python 和 PL/Perl 等，这使得 PostgreSQL 可以结合其他编程语言进行开发和扩展。

## 离线升级中间件 - Postgresql 模块

本页说明从[下载中心](#)下载中间件 - Postgresql 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 \_\_mcamel\_\_ 字样是中间件模块的内部开发代号。

### 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

#### chart-syncer 同步镜像到镜像仓库

1. 创建 load-image.yaml

**!!! note**

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

**==== “已安装 chart repo”**

若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
  url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
````
```

**==== “未安装 chart repo”**

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: LOCAL
  path: ./local-repo # chart 本地路径
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
```

```
password: "Harbor12345" # 你的镜像仓库密码
```

```
```
```

## 2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

### 1. 解压 tar 压缩包。

```
tar -xvf mcamel-postgresql_0.5.1_amd64.tar
cd mcamel-postgresql_0.5.1_amd64
tar -xvf mcamel-postgresql_0.5.1.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

### 2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

### !!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

### ==== “通过 helm repo 升级”

#### 1. 检查 helm 仓库是否存在。

```
```shell
```

```
helm repo list | grep postgresql
````
```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
````
```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-postgresql http://{harbor url}/chartrepo/{project}
````
```

1. 更新 helm 仓库。

```
```shell
helm repo update mcamel/mcamel-postgresql # helm 版本过低会导致失败，若失败，请尝试执行 helm update repo
````
```

1. 选择您想安装的版本（建议安装最新版本）。

```
```shell
helm search repo mcamel/mcamel-postgresql --versions
````
```

```
```none
[root@master ~]# helm search repo mcamel/mcamel-postgresql --versions
NAME          CHART VERSION   APP VERSION   DESCRIPTION
mcamel/mcamel-postgresql    0.5.1        0.5.1        A Helm chart for Kubernetes
...
````
```

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-postgresql -n mcamel-system -o yaml > mcamel-postgresql.yaml
````
```

## 1. 执行 `helm upgrade`。

升级前建议您覆盖 mcamel-postgresql.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-postgresql mcamel/mcamel-postgresql \
-n mcamel-system \
-f ./mcamel-postgresql.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.5.1
```

```

```

==== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-postgresql -n mcamel-system -o yaml > mcamel-postgresql.yaml
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-postgresql . \
-n mcamel-system \
-f ./mcamel-postgresql.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```

```

PostgreSQL Release Notes

本页列出 PostgreSQL 数据库的 Release Notes，便于您了解各版本的演进路径和特性变化。

*[Mccamel-PostgreSQL]: Mccamel 是 DaoCloud 所有中间件的开发代号，PostgreSQL 是一种功能丰富的关系型数据库。

2024-09-30

v0.16.0

- **修复** 选择工作空间查询 PostgreSQL 列表时权限泄漏的问题
- **修复** 部分操作无审计日志的问题

2024-08-31

v0.15.0

- **优化** 创建实例时不可选择异常的集群
- **修复** 纳管 PostgreSQL 实例时内存为空导致失败的问题

2024-06-30

v0.13.0

- **优化** 支持开启 PG Vector 插件
- **优化** PostgreSQL 支持版本升级 至 v15.5

2024-05-31**v0.12.0**

- **新增** 删除任务时可以不删除远程备份
- **新增** 参数模板导入、导出功能
- **优化** 删除备份时可选是否删除 s3 中备份数据
- **优化** 支持批量修改实例参数
- **优化** 支持配置从节点的同步个数

2024-04-30**v0.11.0**

- **优化** 删除备份的时候，可以选择删除在 s3 里的备份数据
- **优化** 增加命名空间配额的提示

2024-03-31**v0.10.0**

- **新增** 参数模板管理
- **新增** 支持通过模板创建 MongoDB 实例
- **修复** 配置访问设置的注释无效的问题

2024-01-31**v0.9.0**

- 优化 在全局管理中增加 PostgreSQL 版本展示

2023-12-31**v0.8.0**

- 优化 监控面板支持中文
- 修复 创建实例时部分输入框填写特殊字符的校验未生效的问题

2023-11-30**v0.7.0**

- 新增 Mcamel-PostgreDB 支持访问白名单设置
- 新增 支持记录操作审计日志
- 优化 实例列表未获取到列表信息时的提示

2023-10-31**v0.6.0**

- 新增 离线升级
- 新增 实例重启功能
- 新增 纳管外部实例

- **修复** Pod 列表展示地址为 Host IP
- **修复** cloudshell 权限问题

2023-08-31

v0.5.0

- **优化** KindBase 语法兼容
- **优化** operator 创建过程的页面展示

2023-07-31

v0.4.0

- **新增** 备份管理能力
- **优化** 监控图表，去除干扰元素并新增时间范围选择
- **修复** 监控图表部分 Panel 无法展示的问题

2023-06-30

v0.3.0

- **新增** 对接全局管理审计日志模块
- **优化** 监控图表，去除干扰元素并新增时间范围选择

2023-04-27**v0.1.2**

- 新增 Mcamel-PostgreSQL UI 模块上线，支持界面化管理
- 新增 安装 Mcamel-PostgreSQL PgAdmin 离线镜像版本能力
- 新增 Mcamel-PostgreSQL PgAdmin 支持 LoadBalancer 类型
- 优化 Mcamel-PostgreSQL 在 IPv6 的情况下 Exporter 没有权限连接到 PostgreSQL
- 优化 Mcamel-PostgreSQL 调度策略增加滑动按钮

2023-04-20**v0.1.1**

- 新增 Mcamel-PostgreSQL 支持 快速启动 PostgreSQL 集群
- 新增 Mcamel-PostgreSQL 支持 ARM 环境下部署
- 新增 Mcamel-PostgreSQL 支持 自定义角色
- 新增 Mcamel-PostgreSQL 支持 PostgreSQL 生命周期管理的后端接口
- 新增 Mcamel-PostgreSQL 支持 查看日志
- 新增 Mcamel-PostgreSQL 支持 PostgreSQL 15
- 新增 Mcamel-PostgreSQL 支持 PostgreSQL UI 管理界面

2023-03-29**v0.0.2**

- 新增 Mcamel-PostgreSQL 支持 PostgreSQL 实例创建

创建 PostgreSQL 实例

接入 PostgreSQL 数据库后，参照以下步骤创建 PostgreSQL 实例。

1. 在实例列表中，点击右上角的 **新建实例**。

基本信息

基本信息

2. 在 **创建 PostgreSQL 实例** 页面中，配置 **基本信息** 后，点击 **下一步**。

基本信息

基本信息

3. 选择部署类型、CPU、内存和存储等 **规格配置** 后，点击 **下一步**。

规格配置

规格配置

!!! info

使用主备模式，更能保证数据的准确性与一致性。

主备部署类型提供高可用，当主节点故障后，备节点自动升级为主节点。包含一个主节点和多个备节点，主备节点的数据通过实时复制保持一致，备节点为只读节点，系统自动进行读请求的负载均衡。

4. 设置用户名和密码等 **服务设置**，默认采用 ClusterIP 作为访问方式。

服务设置

服务设置

5. 支持优化配置 PostgreSQL 的初始化配置，提供了默认通用配置。

服务设置

服务设置

6. 确认基本信息、规格配置、服务设置的信息准确无误后，点击 **确定**。

确认

确认

7. 返回实例列表，屏幕将提示 **创建实例成功**。

成功创建

成功创建

更新 PostgreSQL 实例

如果想要更新或修改 PostgreSQL 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 按钮，在弹出菜单中选择 **更新实例**

更新实例

更新实例

2. 基本信息：只能修改描述。然后点击 **下一步**

基本信息

基本信息

3. 修改规格配置后点击 **下一步**

规格配置

规格配置

4. 修改服务设置后点击 **确定**

服务设置

服务设置

5. 实例配置更新成功，可以在右上角看到更新结果提示。

更新完成

更新完成

!!! note

另外，通过右侧的操作菜单，还可以修改实例的副本数，重启、[删除](./delete.md)实例，配置和管理能够访问实例的白名单。

删除 PostgreSQL 实例

如果想要删除一个 PostgreSQL 实例，可以执行如下操作：

1. 在 PostgreSQL 实例列表中，点击右侧的  按钮，在弹出菜单中选择 **删除实例**

 删除实例

 删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击 **删除** 按钮。

 点击删除

 点击删除

!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

查看 PostgreSQL 日志

操作步骤

通过访问每个 PostgreSQL 的实例详情，页面；可以支持查看 PostgreSQL 的日志。

1. 在 PostgreSQL 实例列表中，选择想要查看的日志，**点击**实例名称进入到实例详情页面。

 image

 image

2. 在实例的左侧菜单栏，会发现有一个 **日志查看** 的菜单栏选项。



3. 点击 **日志查看** 即可进入到日志查看页面 ([Insight 日志查看](#))。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 **自定义日志时间范围**，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 **关键字检索日志**，左侧检索区域支持查看更多的日志信息
- 支持 **日志量分布查看**，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 **查看日志的上下文**，点击右侧 **上下文** 图标即可
- 支持 **导出日志**



手工设置工作负载反亲和

PostgreSQL 中间件的反亲和策略由同一集群下的所有实例共用，因此我们默认开启了 **Preferred** 反亲和。如需关闭反亲和策略或开启 **Required** 反亲和，需要修改 **Operator** 设置。

!!! note

修改 PostgreSQL 的反亲和设置会影响集群内所有 PostgreSQL 实例，建议谨慎操作。

操作步骤

1. 进入 **容器管理 -> 集群列表**，选择实例所在集群；

2. 点击 **自定义资源**，查找资源：**operatorconfigurations.acid.zalan.do**

创建

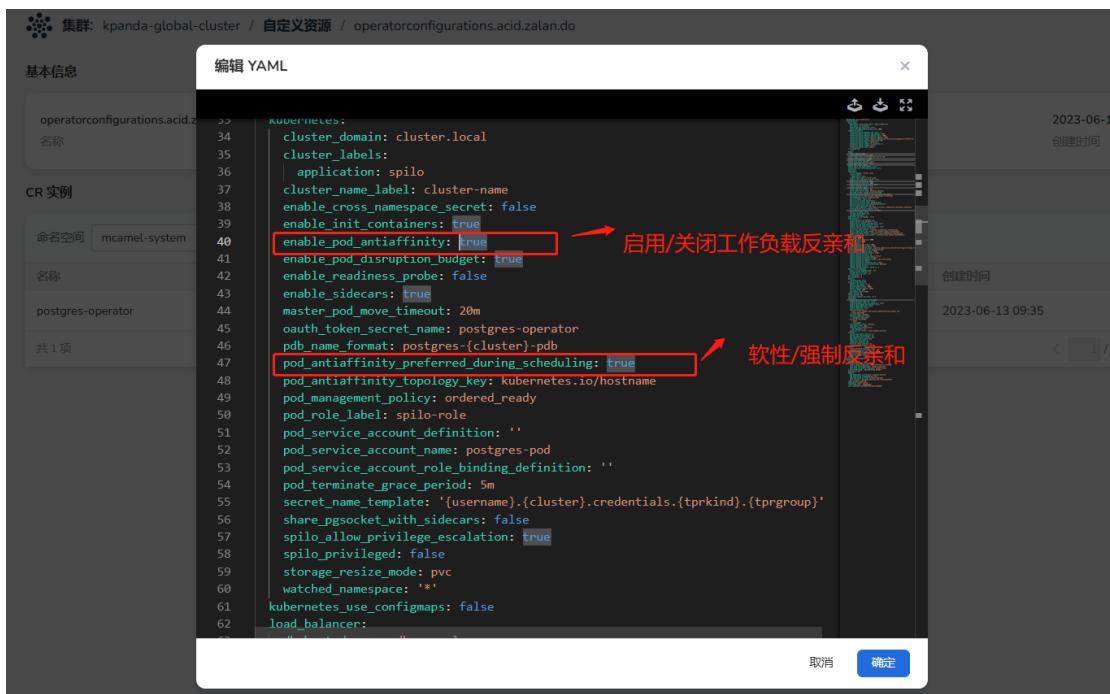
创建

3. 在该资源下选择正确的 **命名空间** -> **CR** 实例

创建

创建

4. 点击 **编辑 YAML**，根据需求修改以下字段



创建

字段

说明

`enable_pod_antiaffinity`

true : 启用工作负载反亲和 false : 关闭工作负载反

亲和

`pod_antiaffinity_preferred_during_sche
duling`

true : Preferred 软性反亲和 false : Required 强制

反亲和

完整代码如下，供参考：

```
apiVersion: acid.zalan.do/v1
configuration:
  aws_or_gcp:
    additional_secret_mount_path: /meta/credentials
    aws_region: eu-central-1
    enable_ebs_gp3_migration: false
    enable_ebs_gp3_migration_max_size: 1000
  connection_pooler:
    connection_pooler_default_cpu_limit: '1'
    connection_pooler_default_cpu_request: 500m
    connection_pooler_default_memory_limit: 100Mi
    connection_pooler_default_memory_request: 100Mi
    connection_pooler_image: registry.opensource.zalan.do/acid/pgbouncer:master-26
    connection_pooler_max_db_connections: 60
    connection_pooler_mode: transaction
    connection_pooler_number_of_instances: 2
    connection_pooler_schema: pooler
    connection_pooler_user: pooler
  crd_categories:
    - all
  debug:
    debug_logging: true
    enable_database_access: true
    docker_image: ghcr.io/zalando/spilo-15:2.1-p9
    enable_crd_registration: true
    enable_crd_validation: true
    enable_lazy_spilo_upgrade: false
    enable_pgversion_env_var: true
    enable_shm_volume: true
    enable_spilo_wal_path_compat: false
    enable_team_id_clusternamespace_prefix: false
    etcd_host: "
  kubernetes:
    cluster_domain: cluster.local
    cluster_labels:
      application: spilo
      cluster_name_label: cluster-name
      enable_cross_namespace_secret: false
      enable_init_containers: true
      enable_pod_antiaffinity: true
      enable_pod_disruption_budget: true
      enable_readiness_probe: false
      enable_sidecars: true
      master_pod_move_timeout: 20m
```

```
oauth_token_secret_name: postgres-operator
pdb_name_format: postgres-{cluster}-pdb
pod_antiaffinity_preferred_during_scheduling: true
pod_antiaffinity_topology_key: kubernetes.io/hostname
pod_management_policy: ordered_ready
pod_role_label: spilo-role
pod_service_account_definition: ""
pod_service_account_name: postgres-pod
pod_service_account_role_binding_definition: ""
pod_terminate_grace_period: 5m
secret_name_template: '{username}.{cluster}.credentials.{tprkind}.{tprgroup}'
share_pgsocket_with_sidecars: false
spilo_allow_privilegeEscalation: true
spilo_privileged: false
storage_resize_mode: pvc
watched_namespace: '*'
kubernetes_use_configmaps: false
load_balancer:
  db_hosted_zone: db.example.com
  enable_master_load_balancer: false
  enable_master_pooler_load_balancer: false
  enable_replica_load_balancer: false
  enable_replica_pooler_load_balancer: false
  external_traffic_policy: Cluster
  master_dns_name_format: '{cluster}.{namespace}.{hostedzone}'
  master_legacy_dns_name_format: '{cluster}.{team}.{hostedzone}'
  replica_dns_name_format: '{cluster}-repl.{namespace}.{hostedzone}'
  replica_legacy_dns_name_format: '{cluster}-repl.{team}.{hostedzone}'
logging_rest_api:
  api_port: 8080
  cluster_history_entries: 1000
  ring_log_lines: 100
logical_backup:
  logical_backup_cpu_limit: 200m
  logical_backup_cpu_request: 100m
  logical_backup_docker_image: registry.opensource.zalan.do/acid/logical-backup:v1.9.0
  logical_backup_job_prefix: logical-backup-
  logical_backup_memory_limit: 256Mi
  logical_backup_memory_request: 128Mi
  logical_backup_provider: s3
  logical_backup_s3_access_key_id: minio
  logical_backup_s3_bucket: poste
  logical_backup_s3_endpoint: http://10.6.216.5:31612
  logical_backup_s3_region: "
```

```
logical_backup_s3_retention_time: 3 days
logical_backup_s3_secret_access_key: Daocloud
logical_backup_s3_sse: ""
logical_backup_schedule: 30 00 * * *
major_version_upgrade:
    major_version_upgrade_mode: 'off'
    minimal_major_version: '12'
    target_major_version: '16'
max_instances: -1
min_instances: -1
patroni:
    failsafe_mode: false
postgres_pod_resources:
    default_cpu_limit: '1'
    default_cpu_request: 100m
    default_memory_limit: 500Mi
    default_memory_request: 100Mi
    min_cpu_limit: 250m
    min_memory_limit: 250Mi
repair_period: 5m
resync_period: 30m
set_memory_request_to_limit: false
teams_api:
    enable_admin_role_for_users: true
    enable_postgres_team_crd: false
    enable_postgres_team_crd_superusers: false
    enable_team_member_deprecation: false
    enable_team_superuser: false
    enable_teams_api: false
    pam_configuration: >-
        https://info.example.com/oauth2/tokeninfo?access_token= uid
        realm=/employees
    pam_role_name: zalandos
    postgres_superuser_teams:
        - postgres_superusers
protected_role_names:
    - admin
    - cron_admin
role_deletion_suffix: _deleted
team_admin_role: admin
team_api_role_configuration:
    log_statement: all
teams_api_url: https://teams.example.com/api/
timeouts:
```

```

patroni_api_check_interval: 1s
patroni_api_check_timeout: 5s
pod_deletion_wait_timeout: 10m
pod_label_wait_timeout: 10m
ready_wait_interval: 3s
ready_wait_timeout: 30s
resource_check_interval: 3s
resource_check_timeout: 10m

users:
  enable_password_rotation: false
  password_rotation_interval: 90
  password_rotation_user_retention: 180
  replication_username: standby
  super_username: postgres

workers: 8

kind: OperatorConfiguration

metadata:
  annotations:
    mcamel/description: cas111
    mcamel/update-timestamp: '2023-07-24T13:27:27+08:00'
    meta.helm.sh/release-name: postgres-operator
    meta.helm.sh/release-namespace: mcamel-system
    creationTimestamp: '2023-06-13T01:35:00Z'
    generation: 10
  labels:
    app.kubernetes.io/instance: postgres-operator
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: postgres-operator
    helm.sh/chart: postgres-operator-0.0.2-553-g4622fd73
  name: postgres-operator
  namespace: mcamel-system
  resourceVersion: '1202085540'
  uid: 3c9a7758-2f76-432e-9f84-0161cf9a959b

```

5. 重启 Operator，已创建的实例也会随之重建并应用新的调度配置。

创建

创建

备份管理

PostgreSQL 支持对运行中的实例数据进行自动或手动备份。

备份配置

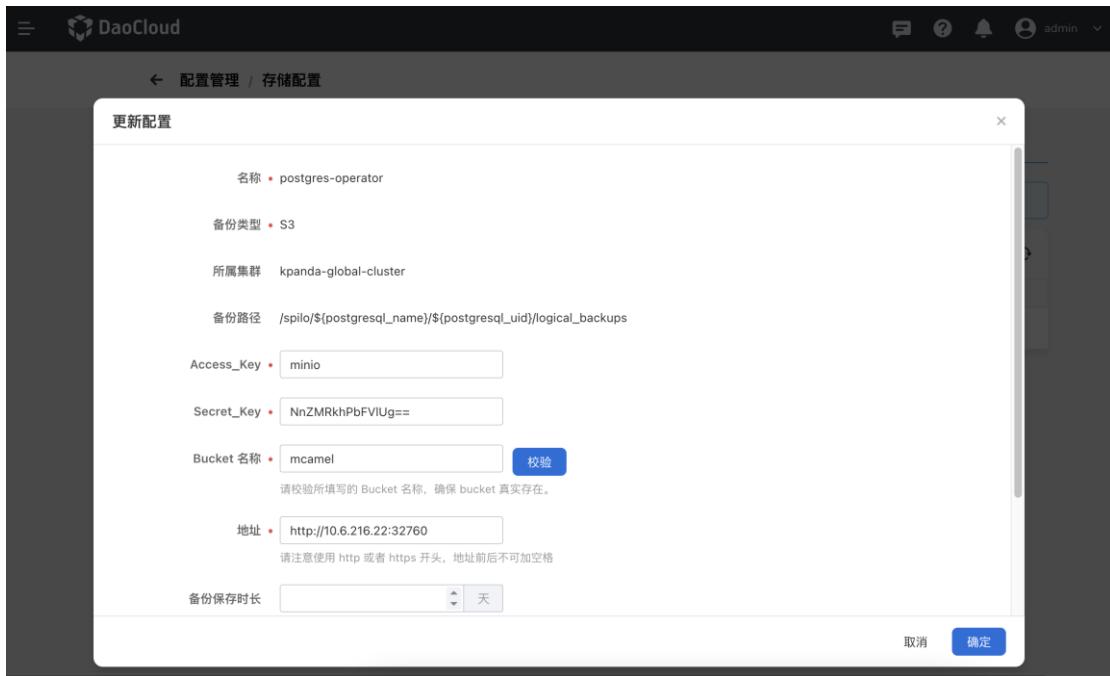
您可以针对不同的实例创建多个备份配置。例如一个实例一个备份配置。

1. 在 PostgreSQL 数据库首页，点击 配置管理

The screenshot shows the PostgreSQL database configuration management interface. At the top, there is a navigation bar with the DaoCloud logo, search, and user information. Below it, a sub-header for 'PostgreSQL 数据库' with a cluster dropdown set to 'default'. A search bar and two buttons ('配置管理' and '新建实例') are also present. The main content area displays a single instance named 'gfre' which is '运行中' (Running). It provides detailed configuration information: deployment location 'kpanda-global-cluster/default', version '15', deployment type '单节点' (Single Node), and replication factor '1 副本'. Resource allocation includes CPU (请求量 0.1 Core, 限制量 1 ...), memory (请求量 0.5 GB, 限制量 1 ...), and disk (1 GB). The status shows '1 / 1 正常/全部副本数'. Below this, a note says '访问地址 gfre-svc.default.svc:5432'. At the bottom, a pagination bar indicates '1 / 1' and '10 项'.

配置管理

2. 在 存储配置 页签中查看已有的配置，或者点击 创建 按钮。



备份配置

- 备份类型：托管 MinIO 或 S3
- 备份实例：选择一个实例
- 输入访问实例的 Access_Key、Secret_Key 和 Bucket 名称

!!! note

在 **参数模板** 页签中，可以创建各种模板用于后续配置。

3. 确认无误后点击 **确定**。

备份设置

备份之前，先要做好备份设置。

进入某个 PostgreSQL 实例，从左侧导航栏点击 **备份管理 -> 备份设置**。



备份设置

自动备份：开启自动备份后，将自动全量备份运行中的实例

创建备份

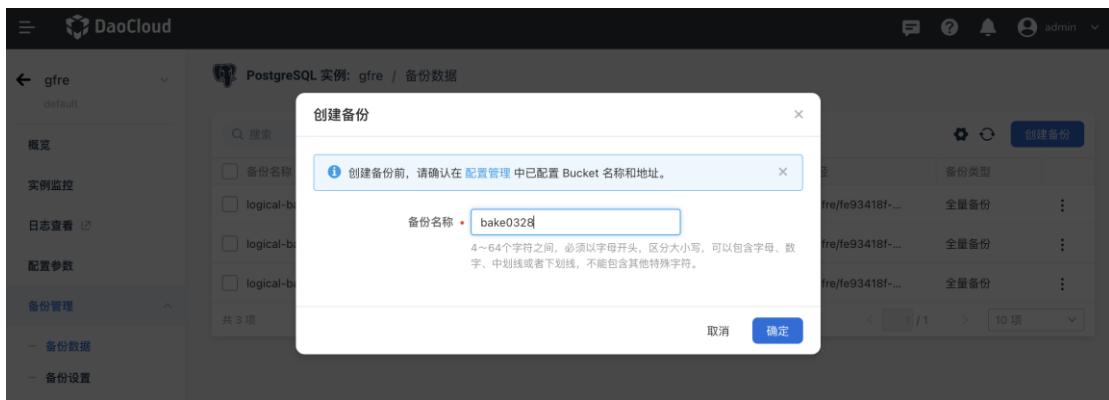
1. 进入某个 PostgreSQL 实例，从左侧导航栏点击 备份管理 -> 备份数据 -> 创建备份

按钮



创建备份

2. 在弹窗中输入一个备份名称后点击 确定



输入名称

3. 屏幕显示创建成功提示，点击右侧的 按钮，可以执行更多操作。

删除备份数据

如果想要删除备份数据，可以在备份数据列表中，点击右侧的 按钮，在弹出菜单中选择

删除。



删除

在弹窗中确认无误后点击 **确定**。

配置备份数据

PostgreSQL 数据库支持数据库实例的自动备份，由于开启备份会损耗数据库读写性能，

因此建议在业务低峰时间段对其自动备份。以保证数据库存在数据丢失时能够快速找回并

保证数据安全。

操作步骤

1. 进入 PostgreSQL 数据库

2. 在实例列表中选择需要开启自动备份的实例，点击其名称进入实例详情。

3. 点击左侧导航栏中的 备份管理 -> 备份数据

| 备份名称 | 备份状态 | 备份方式 | 备份开始时间-备份结束时间 | 备份路径 | 备份类型 |
|----------|------|------|-------------------------------------|--|------|
| full-bak | 备份成功 | 手动 | 2024-08-27 14:53 - 2024-08-27 14:53 | /spilo/pg-harbor/c09702a7-4d77-4c63... | 全量备份 |
| qqwqewq | 备份中 | 手动 | 2024-08-29 10:23 - 暂无数据 | /spilo/pg-harbor/c09702a7-4d77-4c63... | 全量备份 |

auto-backup

创建备份

创建备份前，请确认在 配置管理 中已配置 Bucket 名称和地址。

备份名称 *

4~61个字符之间，只能由小写字母、数字、点(.)和连字符(-)组成，必须以字母、数字开头、结尾。符号(-)不能连续出现。

取消 确定

auto-backup

配置自动备份

备份配置：可设置自动备份，已经自动备份的周期。

- 自动备份：有一个开关，可以选择“启用”或“禁用”自动备份功能。提示信息说明启用后，系统将自动备份所有正在运行的实例。

- **自动备份周期**：可以选择备份的周期，**默认选项为“全选”**，允许用户选择具体的备份星期几（如星期一到星期天）。
- **备份时间段**：允许用户设置备份开始的具体时间，**默认值为 00:00**。



auto-backup

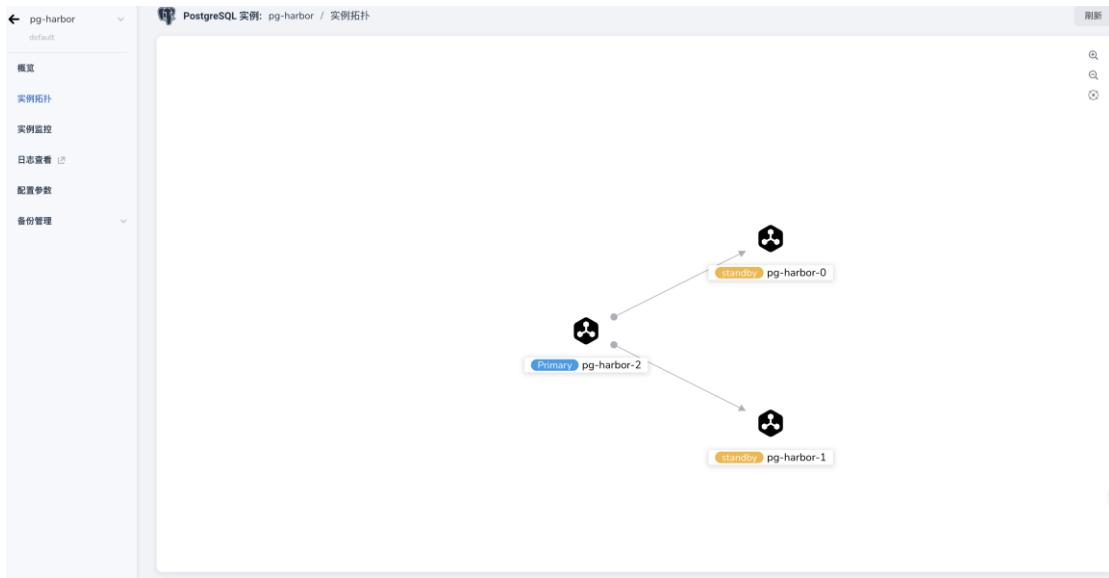
实例拓扑

PostgreSQL 实例拓扑展示实例中节点间数据同步的实例状态，以及基本信息。

操作步骤

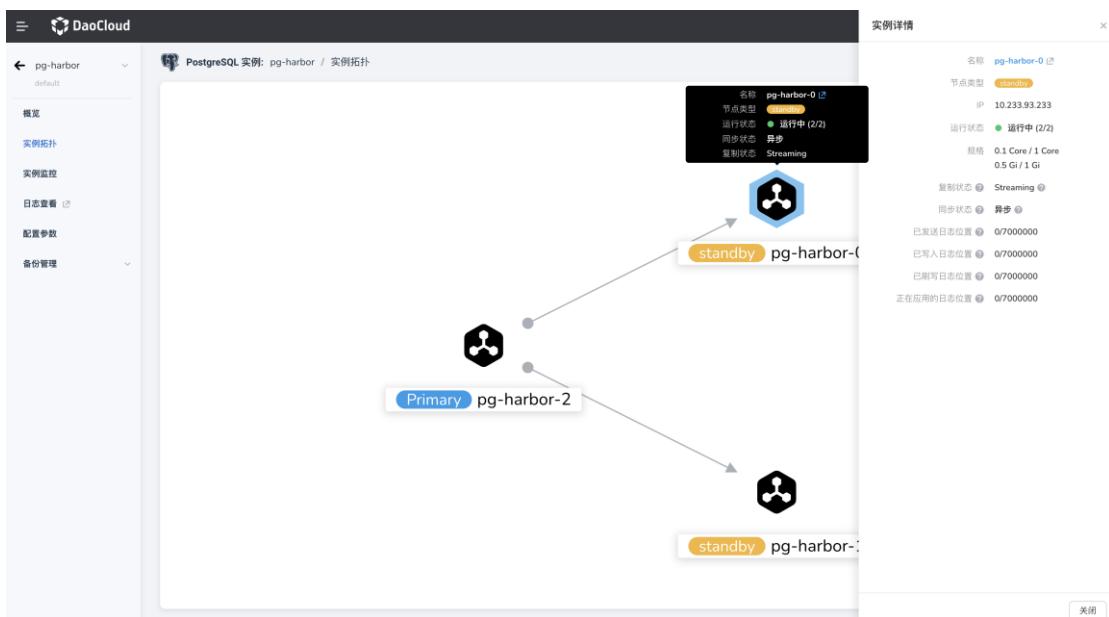
点击进入目标实例的详情页，点击左侧导航栏，选择 **实例拓扑**。

- 在拓扑中，可以看到实例节点之间的主从关系以及数据同步的方向；
- 点击右上角的图标可以放大或缩小拓扑图；



实例拓扑

- 鼠标悬浮在节点上或点击节点时，可以查看所选节点的详细信息，包括其 IP 地址、运行状态、同步状态等。同步状态的信息包括节点的日志状态和同步进度，包括已发送、已写入、已复制的日志量等。



实例拓扑

开启 PGVector 插件

pgvector 是一个在 PostgreSQL 数据库中存储和操作向量数据的扩展，它提供了对高维向量数据的支持，使得用户可以在关系数据库中直接存储、检索和操作向量。它的主要特性包括：

1. 向量存储：支持将高维向量直接存储在 PostgreSQL 表中，方便与其他关系数据结合使用。
2. 相似性搜索：提供了高效的向量相似性搜索算法，如欧氏距离、余弦相似度和内积，使得在数据库中进行向量搜索变得高效和方便。
3. 索引支持：支持使用向量索引（如 L2、IP、Cosine）来加速向量相似性搜索，提高查询性能。

启用 pgvector 扩展

1. 登录到 PostgreSQL 实例，在预启用 pgvector 的数据库中执行以下 SQL 命令，创建 pgvector 扩展插件
- ```
CREATE EXTENSION vector;
```

## 验证 pgvector 插件

1. 创建一个包含向量数据的测试表，并插入一些测试的向量数据。

```
-- 创建测试表
```

```
CREATE TABLE test_vectors (
 id serial PRIMARY KEY,
 embedding vector(3)
);
```

```
-- 插入测试数据
```

```
INSERT INTO test_vectors (embedding) VALUES
('[1, 2, 3]'),
('[4, 5, 6'],
('[7, 8, 9');
```

2. 查询表中的向量数据。

```
SELECT * FROM test_vectors;
```

返回结果如下：

```
```json
id | embedding
---+-----
1 | [1, 2, 3]
2 | [4, 5, 6]
3 | [7, 8, 9]
```

```

## 验证向量相似性搜索

1. 执行以下 SQL，验证相似性搜索功能

```
-- 使用欧氏距离进行相似性搜索
SELECT id, embedding
FROM test_vectors
ORDER BY embedding <-> '[1, 2, 3]'
LIMIT 5;
```

返回结果如下：

```
```json
id | embedding
---+-----
1 | [1, 2, 3]
2 | [4, 5, 6]
3 | [7, 8, 9]
```

```

## 什么是 Redis 缓存服务

Redis 缓存服务是 DaoCloud 提供的一款内存数据库缓存服务，兼容了 Redis 和 Memcached 两种内存数据库引擎，为您提供开箱即用、安全可靠、弹性扩容、便捷管理的

在线分布式缓存能力，满足用户高并发及数据快速访问的业务诉求。

- **开箱即用**

提供单机、高可用集群、Cluster 集群、读写分离类型的缓存实例，拥有从 128M 到 1024G 的丰富内存规格。您可以通过 UI 控制台直接创建，无需单独准备服务器资源。

所有 Redis 版本采用容器化部署，秒级完成创建。

- **安全可靠**

借助 DaoCloud 全局管理、审计日志等安全管理服务，全方位保护实例数据的存储与访问。

灵活的容灾策略，主备/集群实例从单集群内部署，到支持多集群多云部署。

- **弹性伸缩**

提供对实例内存规格的在线扩容与缩容服务，帮助您实现基于实际业务量的成本控制，达到按需使用的目标。

- **便捷管理**

可视化 Web 管理界面，在线完成实例重启、参数修改等操作。还提供 RESTful API，方便您进一步实现实例自动化管理。

[创建 Redis 实例](#)

## 适用场景

很多大型电商网站、视频直播和游戏平台等，存在 **大规模数据访问**，对 **数据查询效率**要求高，且 **数据结构简单，不涉及太多关联查询**。这种场景使用 Redis，在速度上对传统磁盘数据库有很大优势，能够有效减少数据库磁盘 IO，提高数据查询效率，减轻管理维护

工作量，降低数据库存储成本。Redis 对传统磁盘数据库是一个重要的补充，成为了互联网应用，尤其是支持高并发访问的互联网应用必不可少的基础服务之一。

以下举几个典型样例：

#### 1. 电商网站 - 秒杀抢购

电商网站的商品类目、推荐系统以及秒杀抢购活动，适宜使用 Redis 缓存数据库。例如秒杀抢购活动，并发高，对于传统关系型数据库来说访问压力大，需要较高的硬件配置（如磁盘 IO）支撑。Redis 数据库单节点 QPS 支撑能达到 10 万，轻松应对秒杀并发。实现秒杀和数据加锁的命令简单，使用 SET、GET、DEL、RPUSH 等命令即可。

#### 2. 视频直播 - 消息弹幕

直播间的在线用户列表，礼物排行榜，弹幕消息等信息，都适合使用 Redis 中的 SortedSet 结构进行存储。例如弹幕消息，可使用 ZREVRANGEBYSCORE 排序返回，在 Redis 5.0 中，新增了 zpopmax、zpopmin 命令，更加方便消息处理。

#### 3. 游戏应用 - 游戏排行榜

在线游戏一般涉及排行榜实时展现，比如列出当前得分或战力最高的 10 个用户。使用 Redis 的有序集合存储用户排行榜非常合适，有序集合使用非常简单，提供多达 20 个操作集合的命令。

#### 4. 社交 App - 返回最新评论/回复

在 web 类应用中，常有“最新评论”之类的查询，如果使用关系型数据库，往往涉及到按评论时间逆排序，随着评论越来越多，排序效率越来越低，且并发频繁。使用 Redis 的 List（链表）来存储最新 1000 条评论，当请求的评论数在这个范围，

就不需要访问磁盘数据库，直接从缓存中返回，减少数据库压力的同时，提升 App 的响应速度。

# 离线升级中间件 - Redis 模块

本页说明从[下载中心](#)下载中间件 - Redis 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `_mcamel_` 字样是中间件模块的内部开发代号。

## 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 `chart-syncer` 同步镜像到镜像仓库，该方法更加高效便捷。

### chart-syncer 同步镜像到镜像仓库

#### 1. 创建 `load-image.yaml`

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

==== “已安装 chart repo”

若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mc当地-offline # 到执行 charts-syncer 命令的相对路径,
  而不是此 YAML 文件和离线包之间的相对路径
  target:
    containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
    containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
    repo:
      kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
      url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
      auth:
```

```

username: "admin" # 你的镜像仓库用户名
password: "Harbor12345" # 你的镜像仓库密码
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

==== “未安装 chart repo”

若当前环境未安装 chart repo, chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: LOCAL
  path: ./local-repo # chart 本地路径
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```

tar -xvf mcamel-redis_0.11.1_amd64.tar
cd mcamel-redis_0.11.1_amd64
tar -xvf mcamel-redis_0.11.1.bundle.tar

```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar

- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

**!!! note**

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

==== “通过 helm repo 升级”

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep redis
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-redis http://{harbor url}/chartrepo/{project}
```

```

1. 更新 helm 仓库。

```
```shell
helm repo update mcamel/mcamel-redis # helm 版本过低会导致失败，若失败，请尝试执行 helm update repo
```

```

三

1. 选择您想安装的版本（建议安装最新版本）。

``shell

```
helm search repo mcamel/mcamel-redis --versions
```

三

``none

```
[root@master ~]# helm search repo mcamel/mcamel-redis --versions
```

| NAME                | CHART VERSION | APP VERSION | DESCRIPTION                 |
|---------------------|---------------|-------------|-----------------------------|
| mcamel/mcamel-redis | 0.11.1        | 0.11.1      | A Helm chart for Kubernetes |
| ...                 | ...           | ...         | ...                         |
| ...                 | ...           | ...         | ...                         |

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的`--set`参数。

``shell

```
helm get values mcamel-redis -n mccamel-system -o yaml > mccamel-redis.yaml
```

1. 执行 `helm upgrade` 。

升级前建议您覆盖 `mcamel-redis.yaml` 中的 `'global.imageRegistry'` 字段为当前使用的镜像仓库地址。

``shell

```
export imageRegistry={你的镜像仓库}
```

三

``shell

```
helm upgrade mcamel-redis mcamel/mcamel-redis \
```

-n mcamel-system \

```
-f ./mcamel-redis.yaml \
```

```
--set global.imageRegistry=$imageRegistry \
```

--version 0.11.1

三

==== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-redis -n mccamel-system -o yaml > mccamel-redis.yaml
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```

```
```shell
helm upgrade mcamel-redis . \
-n mccamel-system \
-f ./mcamel-redis.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```

```

## Redis 缓存服务 Release Notes

本页列出 Redis 缓存服务的 Release Notes，便于您了解各版本的演进路径和特性变化。

\*[mcamel-redis]: mccamel 是 DaoCloud 所有中间件的开发代号，redis 是提供内存数据库缓存服务的中间件

**2025-02-28**

**v0.27.0**

- **新增** 支持创建 7.2.7 版本的 Redis 实例

**2024-11-30****v0.24.0**

- **优化** 支持部署 7.2.6 版本的 Redis 实例
- **优化** Redis 哨兵模式支持支持设置非动态参数
- **修复** Redis 哨兵模式重启后无法启动的问题

**2024-09-30****v0.22.0**

- **修复** 选择工作空间查询 Redis 列表时权限泄漏的问题
- **修复** 部分操作无审计日志的问题

**2024-08-31****v0.21.0**

- **优化** 创建实例时不可选择异常的集群

**2024-07-31****v0.20.0**

- **新增** 支持纳管 Redis 实例

## 2024-06-30

### v0.19.0

- 优化 创建 Redis 实例时支持配置固定 IP
- 优化 参数模板增加 \*\*\*\*maxclients 等常用参数

## 2024-05-31

### v0.18.0

- 新增 删除任务时可以不删除远程备份
- 新增 参数模板导入功能
- 优化 支持批量修改实例参数

## 2024-04-30

### v0.17.0

- 新增 Redis 实例拓扑
- 优化 增加命名空间配额的提示
- 优化 删除备份的时候，可以选择删除在 s3 里的备份数据

## 2024-03-31

### v0.16.0

- 新增 Redis 慢查询

- 优化 Redis 哨兵模式支持开启 hostnetwork
- 优化 当用户权限不足时无法读取 redis 的密码
- 优化 升级 Redis 哨兵模式的 Operator
- 修复 Redis 哨兵模式无法创建副本数大于 3 的问题 (需升级 Operator)
- 修复 Redis 哨兵模式宕机 Master 后用户侧有概率不可用的问题 (需升级 Operator)
- 修复 备份路径默认为 name.rdb
- 修复 Redis 哨兵模式监控面板内存使用率有误的问题

## 2024-01-31

### v0.15.0

- 优化 Redis Operator 升级版本至 6.2.12
- 优化 在全局管理中增加 Redis 版本展示
- 修复 Redis 集群模式扩容不生效问题 (需升级 Operator)
- 修复 Redis 集群模式支持 NodePort 访问 (需升级 Operator)

## 2023-12-31

### v0.14.0

- 新增 Redis 内置告警规则
- 优化 Redis 哨兵模式下 NodePort 模式时返回多个哨兵的 NodePort 地址
- 修复 创建实例时部分输入框填写特殊字符的校验未生效的问题

**2023-11-30****v0.13.0**

- **新增** 支持记录操作审计日志
- **优化** 实例列表未获取到列表信息时的提示
- **优化** Mcamel-Redis 监控 Dashboard 的中英文展示

**2023-10-31****v0.12.0**

- **新增** 离线升级
- **新增** 实例重启功能
- **新增** 参数模板功能
- **新增** 跨集群恢复实例
- **优化** 主从延迟的计算方式
- **修复** cloudshell 权限问题

**2023-08-31****v0.11.0**

- **优化** KindBase 语法兼容
- **优化** operator 创建过程的页面展示

## 2023-07-31

### v0.10.0

- 新增 **mcamel-redis** 访问白名单配置
- 优化 **mcamel-redis** 创建实例对话框添加默认反亲和标签值，简化配置过程
- 优化 **mcamel-redis** 数据恢复界面
- 优化 **mcamel-redis** 前端界面权限展示信息
- 修复 **mcamel-redis** 关闭节点亲和性失败

## 2023-06-30

### v0.9.0

- 新增 **mcamel-redis** 不同命名空间下无法创建同名 Redis
- 新增 **mcamel-redis** 非 MCamel 纳管的 Redis 可能被误操作的问题
- 优化 **mcamel-redis** 备份管理页面结构样式展示
- 优化 **mcamel-redis** 备份 Job 中的密码展示问题
- 优化 **mcamel-redis** 监控图表，去除干扰元素并新增时间范围选择
- 优化 **mcamel-redis** ServiceMonitor 闭环安装

## 2023-05-30

### v0.8.0

- 新增 **mcamel-redis** 可配置实例反亲和性
- 新增 **mcamel-redis** 对接全局管理审计日志模块

- 修复 mcamel-redis 删除 Redis 后，备份相关内容残留
- 修复 mcamel-redis 哨兵集群的 Service 地址展示错误

## 2023-04-27

### v0.7.1

- 新增 mcamel-redis 详情页面展示相关的事件
- 新增 mcamel-redis 列表接口支持 Cluster 与 Namespace 字段过滤
- 新增 mcamel-redis 自定义角色
- 修复 mcamel-redis 优化 调度策略增加滑动按钮

## 2023-03-29

### v0.6.2

- 新增 mcamel-redis 支持自动化备份恢复
- 修复 没有导出备份恢复的离线镜像
- 修复 mcamel-redis 没有导出备份恢复的离线镜像
- 修复 mcamel-redis 修复了若干已知问题，提升了系统稳定性和安全性
- 新增 新增备份功能使用文档

## 2023-02-23

### v0.5.0

- 新增 mcamel-redis helm-docs 模板文件

- **新增** `mcamel-redis` 应用商店中的 Operator 只能安装在 `mcamel-system`
- **新增** `mcamel-redis` 支持 cloud shell
- **新增** `mcamel-redis` 支持导航栏单独注册
- **新增** `mcamel-redis` 支持查看日志
- **新增** `mcamel-redis` 更新单例/集群模式 Operator 的版本
- **新增** `mcamel-redis` 展示 common Redis 集群
- **新增** `mcamel-redis` Operator 对接 chart-syncer
- **修复** `mcamel-redis` 实例名太长导致自定义资源无法创建的问题
- **修复** `mcamel-redis` 工作空间 Editor 用户无法查看实例密码
- **修复** `mcamel-redis` 不能解析出正确的 Redis 版本号
- **修复** `mcamel-redis` 无法修改 Port 的问题
- **升级** `mcamel-redis` 升级离线镜像检测脚本
- **新增** 日志查看操作说明，支持自定义查询、导出等功能

**2022-12-25**

**v0.4.0**

- **新增** `mcamel-redis` NodePort 端口冲突提前检测
- **新增** `mcamel-redis` 节点亲和性配置
- **修复** `mcamel-redis` 单例和集群设置 nodeport 无效的问题
- **修复** `mcamel-redis` 集群模式下，从节点不可以设置为 0 的问题

**2022-11-28****v0.2.6**

- **修复** 更新 Redis 时校验部分字段错误
- **改进** 密码校验调整为 MCamel 低等密码强度
- **改进** 提升哨兵模式的版本依赖，v1.1.1=>v1.2.2，重要变更为支持 k8s 1.25+
- **新增** 支持在 ARM 环境安装主备模式的 Redis 集群
- **新增** sc 扩容提示
- **新增** 返回列表或者详情时的公共字段
- **新增** 增加返回告警列表
- **新增** 校验 Service 注释
- **修复** 服务地址展示错误

**2022-10-26****v0.2.2**

- **新增** 获取用户列表接口
- **新增** 支持 arm 架构
- **新增** redis 实例全生命周期的管理
- **新增** redis 实例的监控部署
- **新增** 支持 redis 哨兵，单例和集群一键部署
- **新增** 支持 ws 权限隔离
- **新增** 支持在线动态扩容

- 升级 release notes 脚本

# 创建 Redis 实例

接入 Redis 缓存服务后，参照以下步骤创建 Redis 实例。

1. 在 Redis 缓存服务的实例列表中，点击 新建实例 按钮。

创建按钮

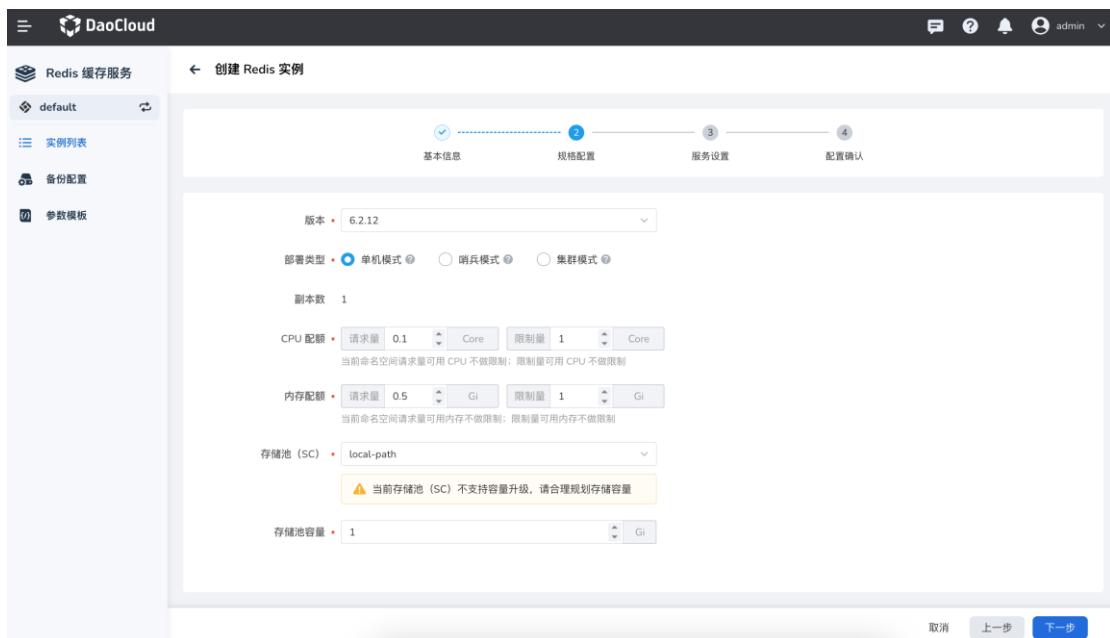
2. 在 创建 Redis 实例 页面中，配置 基本信息 后，点击 下一步

基本信息

3. 选择版本、部署类型、CPU、内存和存储等 规格配置 后，点击 下一步 。 部署类型

有三种模式：

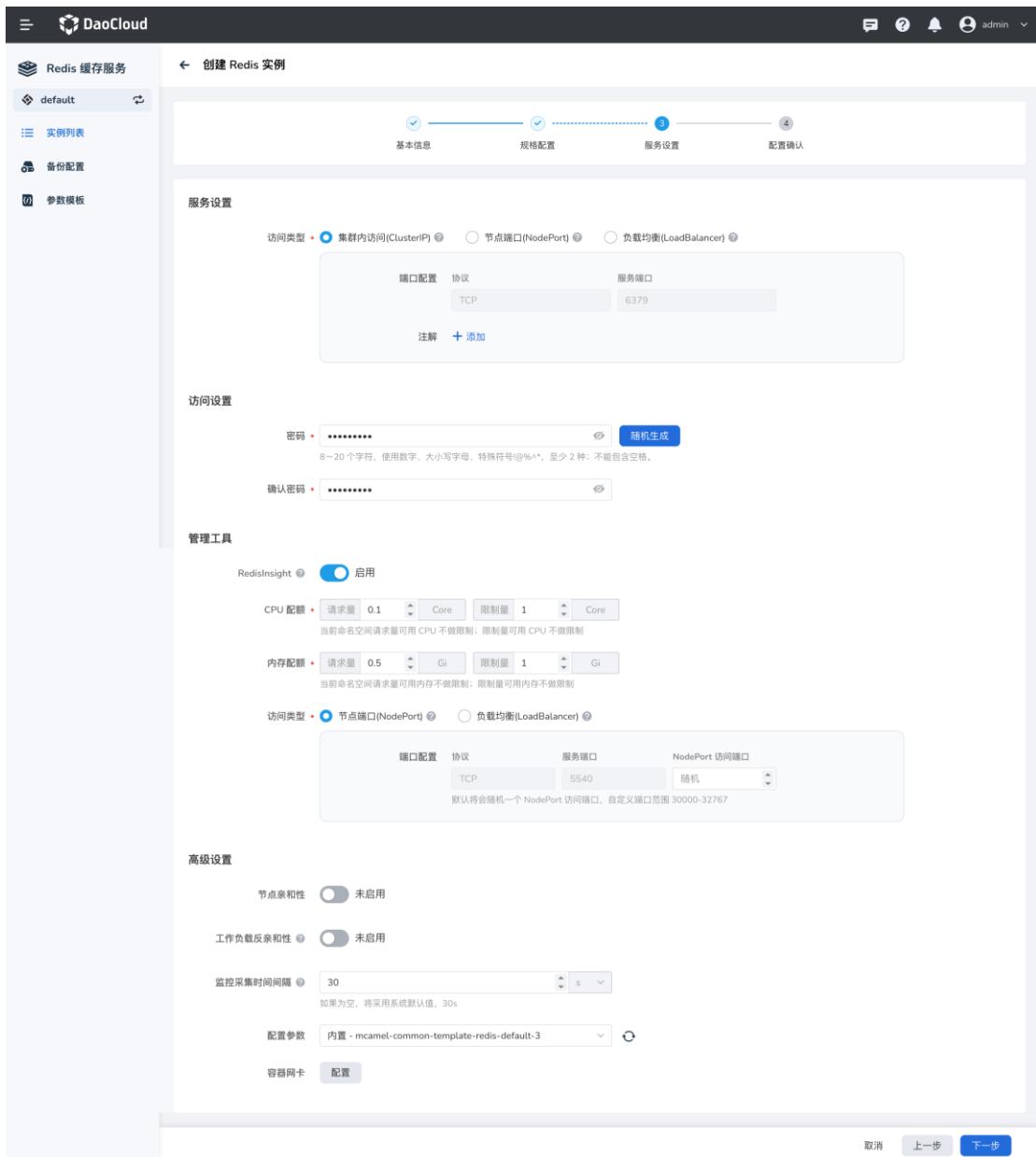
- **单机模式**：仅有主节点。当出现故障时，无法保障数据可靠性。
- **哨兵模式**：当主节点故障后，备节点自动升级为主节点。可保证服务的高可用。
- **集群模式**：支持多主多从结构。可满足用户低延迟、高性能的业务需求。



## 规格配置

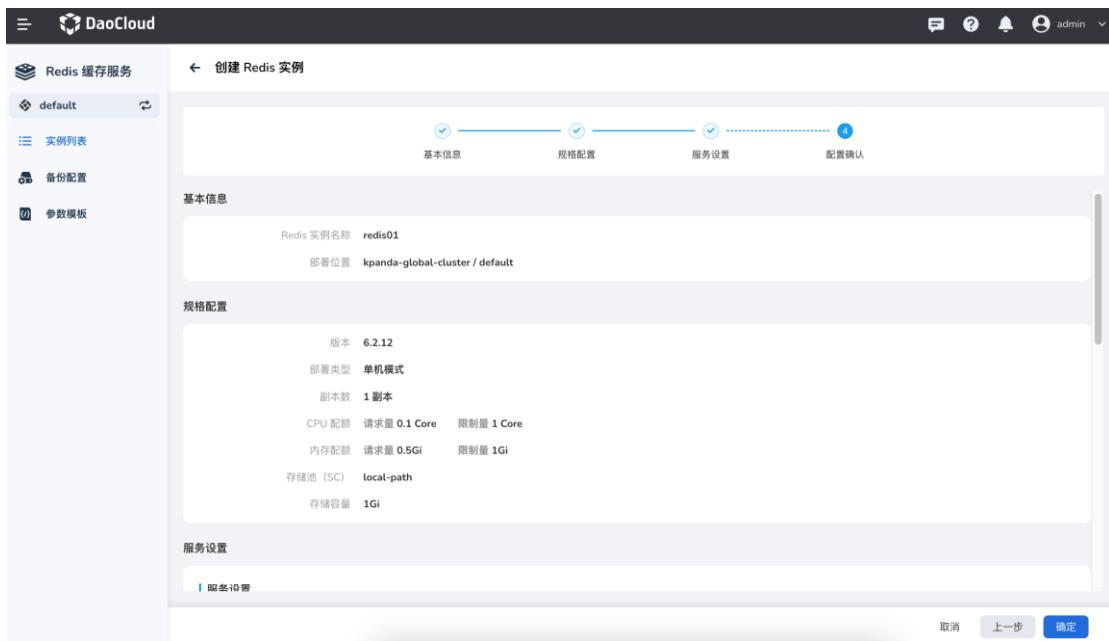
### 4. 设置用户名和密码等 服务设置

- **服务设置**：选择一种访问类型，默认为 ClusterIP
- **访问设置**：随机生成密码，或手动设置密码
- **管理工具**：可选择启用 RedisInsight，这是一款管理和监控工具，能够提供界面化的 Redis 的使用和管理
- **高级设置**：



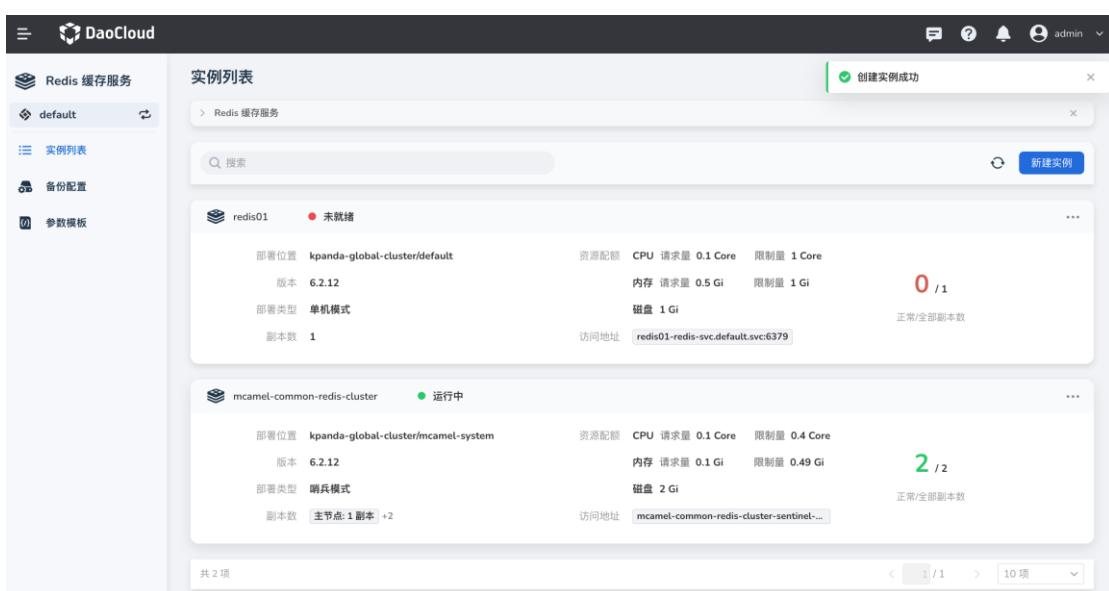
## 服务设置

5. 确认基本信息、规格配置、服务设置的信息准确无误后，点击 **确定**



确认

6. 返回实例列表，屏幕将提示 **创建实例成功**。新创建的实例状态为 **未就绪**，等待片刻后将变为 **运行中**。



成功创建

## 更新 Redis 实例

如果想要更新或修改 Redis 的资源配置，可以按照本页说明操作。

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择 **更新实例**

The screenshot shows the Redis instance list. An instance named 'mcamel-common-redis-cluster' is selected, showing its details: deployment location 'kpanda-global-cluster/mcamel-system', version '6.2.12', deployment mode '哨兵模式' (Sentinel), and replication factor '主节点: 1 副本 +2'. On the right, a context menu is open with several options: '更新实例' (selected), '更新副本数', '访问白名单(0)', '重启', and '删除实例'.

### 更新实例

2. 基本信息：只能修改描述。然后点击 **下一步**

The screenshot shows the 'Basic Information' step of the update wizard. The 'Redis 实例名称' (Instance Name) is set to 'mcamel-common-redis-cluster'. The '描述' (Description) field contains the same name. Below it, the '部署位置' (Deployment Location) section shows '集群: kpanda-global-cluster' and '命名空间: mcamel-system'. At the bottom right, there are '取消' (Cancel) and '下一步' (Next Step) buttons.

### 基本信息

3. 修改规格配置后点击 **下一步**

The screenshot shows the DaoCloud Enterprise 5.0 web interface for managing a Redis cache service. The top navigation bar includes the DaoCloud logo, user information (admin), and a search bar. The left sidebar has a tree view with 'Redis 缓存服务' selected, showing 'default' and '实例列表'. Below the sidebar are '备份配置' and '参数模板' buttons.

The main content area is titled '更新实例 mcamel-common-redis-cluster' and shows a three-step process: '基本信息' (Step 1), '规格配置' (Step 2, currently active), and '服务设置' (Step 3). The '基本信息' step shows the instance version as 6.2.12 and deployment type as '哨兵模式'.

**主从节点**: Shows a '副本数' (Replica Count) of 2. A note states: '哨兵模式下，其中一个副本为主节点，其他副本为从节点。' Configuration for CPU and memory is shown with '请求量' (Request) and '限制量' (Limit) fields for both cores and gigabytes.

**存储池 (SC)**: Set to 'local-path' with a capacity of 2Gi.

**哨兵节点**: Shows a '副本数' (Replica Count) of 1. Configuration for CPU and memory is shown with '请求量' (Request) and '限制量' (Limit) fields for both cores and gigabytes.

At the bottom right are '取消' (Cancel), '上一步' (Previous Step), and a blue '下一步' (Next Step) button.

## 规格配置

4. 修改服务设置后点击 确定

## 服务设置

# 删除 Redis 实例

如果想要删除一个 Redis 实例，可以执行如下操作：

- 在 Redis 实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择 **删除实例**。

## 选择删除实例

2. 在弹窗中输入该实例的名称，确认无误后，点击 **删除** 按钮。



点击删除

**!!! warning**

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

## Redis 手动切换主备

在 Redis 缓存服务中，通过手动切换主从角色，可以在主节点发生故障或需要维护时，确保系统的持续运行。本文将介绍如何通过 UI 手动切换 Redis 的主从角色，并讨论使用场景和限制。

### 使用场景

**1. 故障恢复：**当主节点出现故障时，可以将从节点提升为新的主节点，以确保服务的连续性。

**2. 负载均衡：**在高负载情况下，可以通过切换主从角色来分散请求，提升系统的响应能力。

**3. 测试环境：**在开发和测试环境中，手动切换主从角色可以帮助开发人员验证系统的容

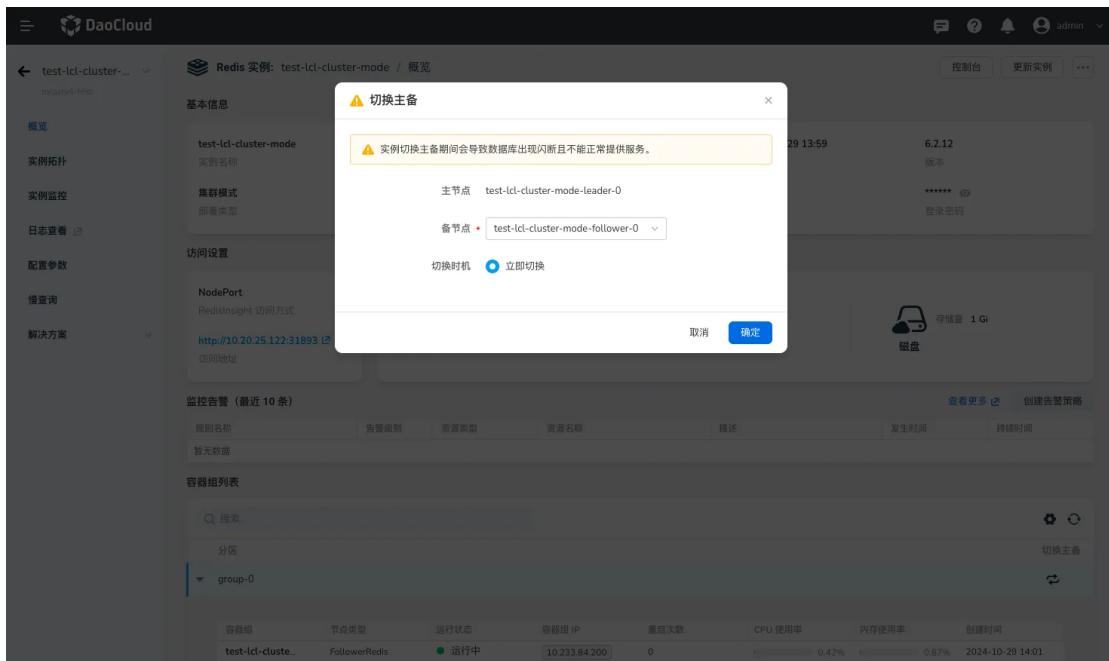
错能力和数据一致性。

## 集群模式

1. 点击目标 Redis 实例的名称，进入其详情页。
2. 在需要进行主备切换的实例的容器组列表中，点击分片名称前面的图标，展开当前分片下所有节点。
3. 如需对某一组分片中的主备节点角色进行切换，可点击分片后的 **切换主从** 图标，将备节点升级为主节点。

{: width=}

4. 展开备节点的下拉框可以指定升级为主节点的备节点。

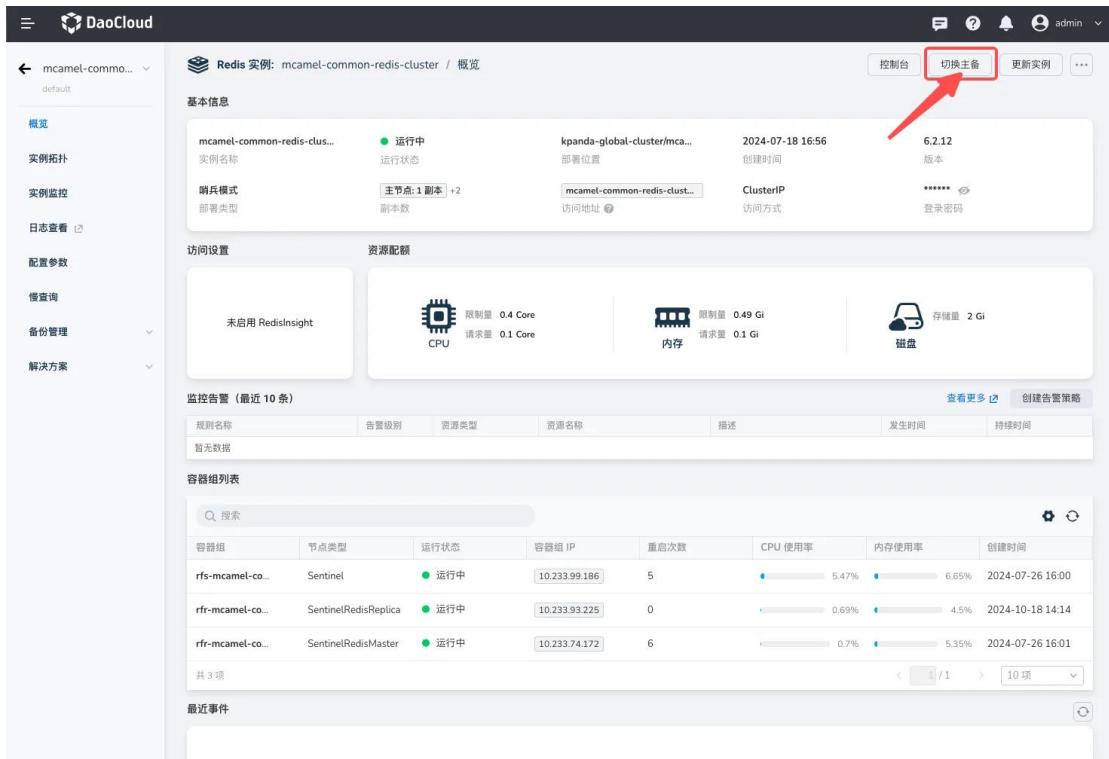


switch-role

## 哨兵模式

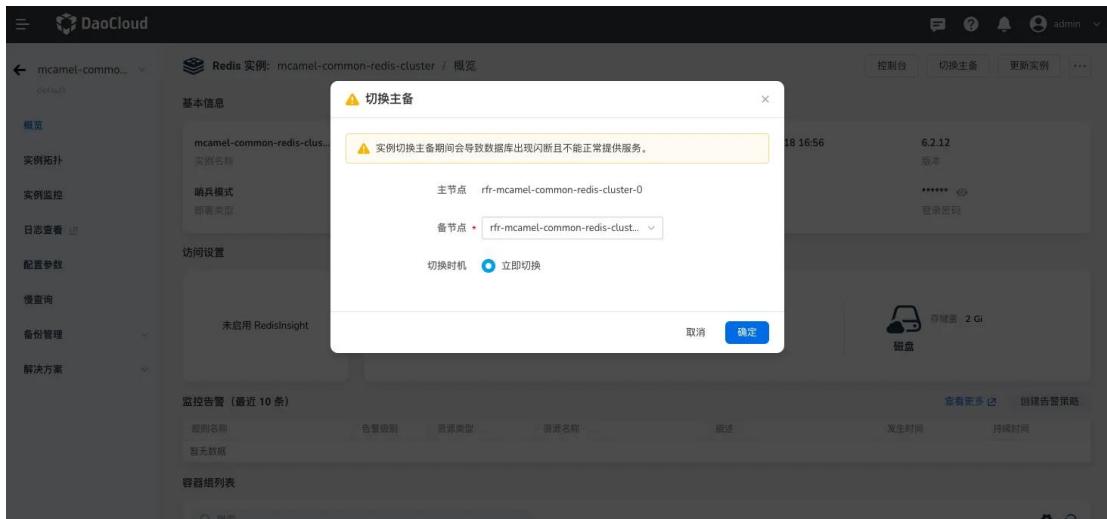
1. 点击目标 Redis 实例的名称，进入其详情页

2. 在概览页的右上角，点击 切换主备



switch-role

3. 展开备节点的下拉框，可以指定升级为主节点的备节点



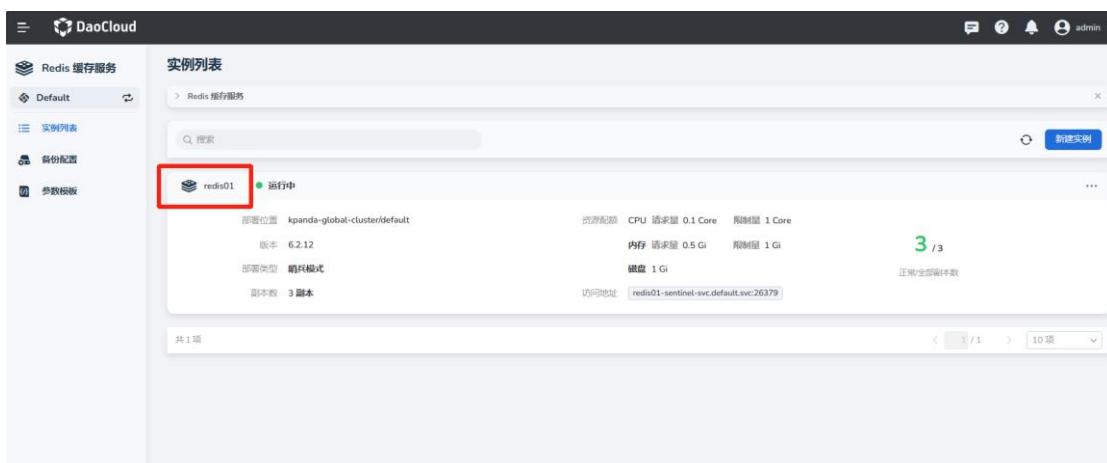
switch-role

## 查看 Redis 日志

### 操作步骤

通过访问每个 Redis 的实例详情页面；可以支持查看 Redis 的日志。

1. 在 Redis 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。



image

2. 在实例的左侧菜单栏，点击 **日志查看** 的菜单栏选项，即可进入到日志查看页面

( [Insight](#) 日志查看 )。

The screenshot shows the Redis instance details page in the DaoCloud Enterprise 5.0 interface. The left sidebar has a 'Logs' tab highlighted with a red box. The main content area includes:

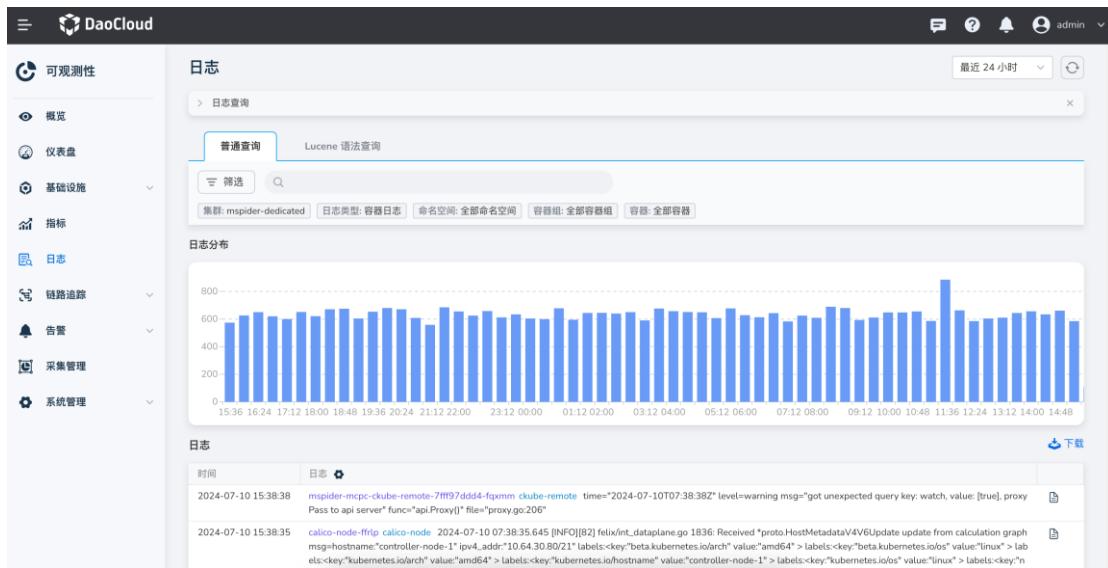
- 基本信息**: Shows the instance name (redis01), status (运行中), deployment (kpanda-global-cluster/default), creation time (2024-07-10 14:31), version (6.2.12), and cluster ID (ClusterIP).
- 资源配额**: Displays CPU (1 Core), Memory (1 Gi), and Disk (1 Gi) usage.
- 监控告警 (最近 10 条)**: A table showing monitoring alerts with columns: 规则名称, 告警级别, 资源类型, 资源名称, 描述, 发生时间, 持续时间. It shows '暂无数据'.
- 容器组列表**: A table listing container groups with columns: 容器组, 运行状态, 容器组 IP, 重启次数, CPU 使用率, 内存使用率, 创建时间. Two entries are shown: rfs-redis01-55fb4d78... (运行中, 10.233.119.234, 0, 0%, 0%, 2024-07-10 14:31) and rfs-redis01-55fb4d78... (运行中, 10.233.127.56, 0, 0%, 0%, 2024-07-10 14:31).

image

## 日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围  
( 可查看的日志范围以 可观测系统设置内保存的日志时长为准 )
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 **上下文** 图标即可
- 支持 导出日志



image

## 慢查询

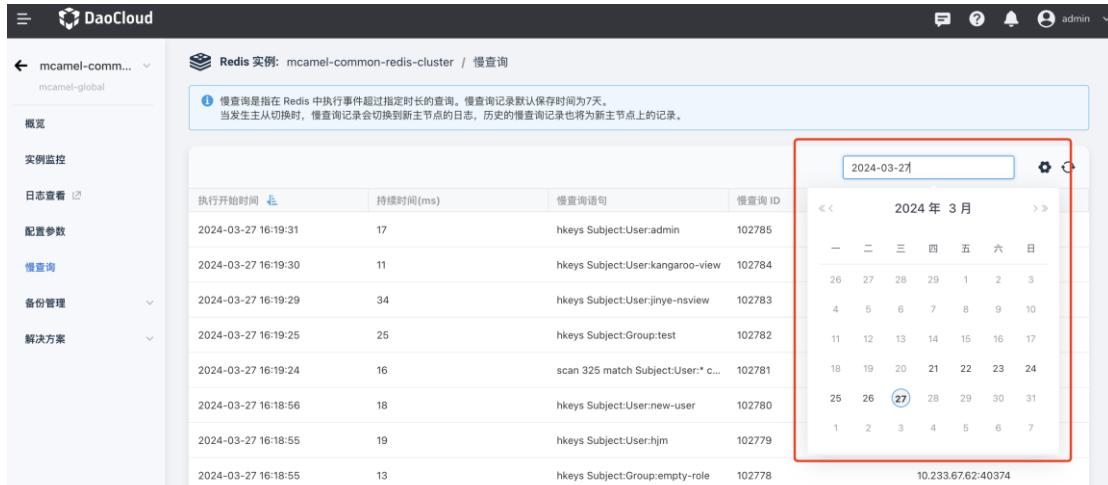
慢查询是指在 Redis 中执行事件超过指定时长的查询。慢查询记录默认保存时间为 7 天。

当发生主从切换时，慢查询记录会切换到新主节点的日志，历史的慢查询记录也将是新主

节点上的记录。

除了传统的命令行外，DCE 5.0 Redis 提供了图形界面来进行慢查询。

您可以查询 7 天中任意一天内执行时长超过预设值的事件。



slow query

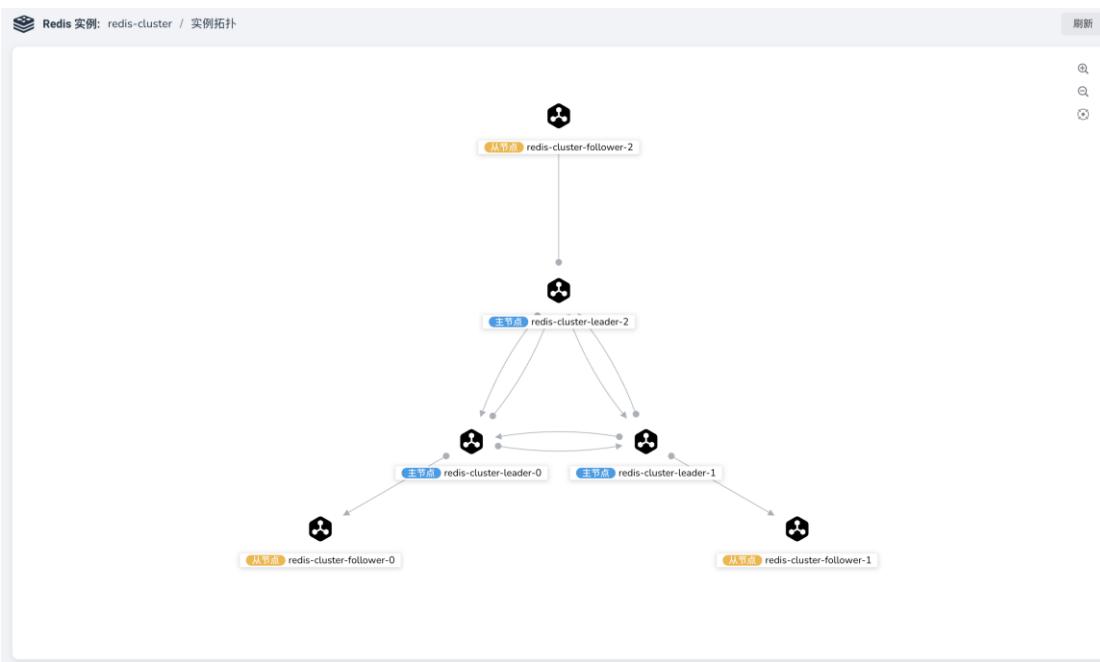
# 实例拓扑

Redis 实例拓扑展示实例中节点间数据同步的实例状态，以及基本信息。

## 操作步骤

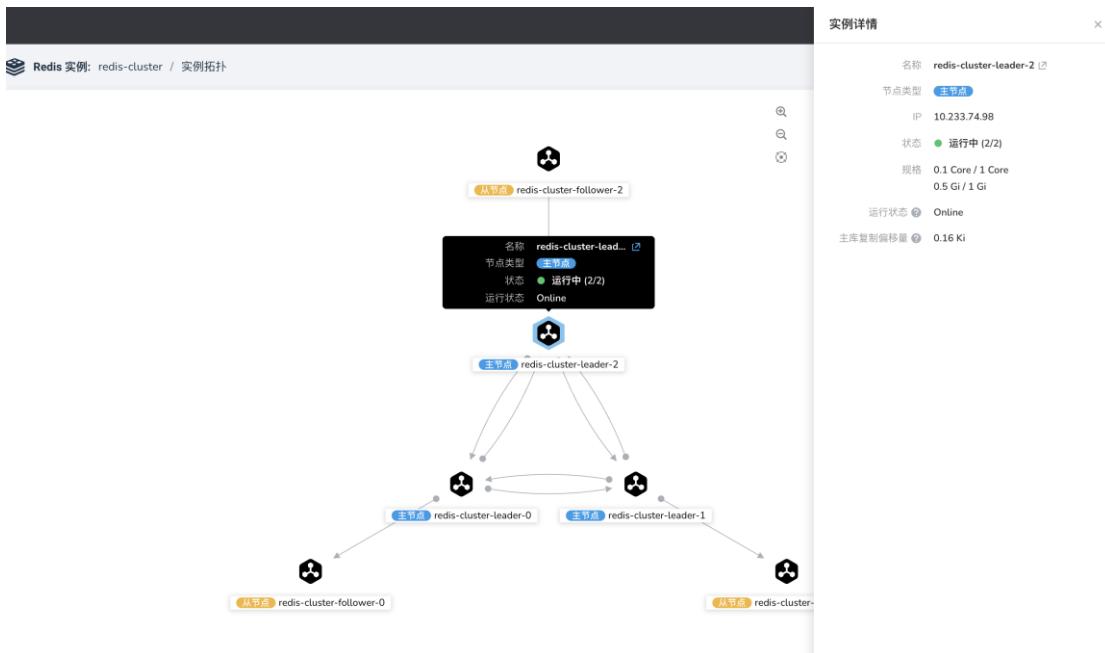
点击进入目标实例的详情页，点击左侧导航栏，选择 实例拓扑。

- 在拓扑中，可以看到实例节点之间的主从关系以及数据同步的方向；
- 点击右上角的图标可以放大或缩小拓扑图；



实例拓扑

- 鼠标悬浮在节点上或点击节点时，可以查看所选节点的详细信息，包括名称、其 IP 地址、运行状态、规格、主库复制量。



实例拓扑

## Redis 备份还原

Redis 会在指定的时间间隔内自动生成数据快照，并将其保存为 .rdb 文件。Redis 的备份与还原功能可以确保数据持久性和安全性。本文将介绍如何执行 Redis 缓存服务数据的备份与还原操作。

### 前提条件

在开始给 Redis 实例备份前，请确认当前工作空间的备份配置中已有配置验证过的 S3 存储。

| 配置名称          | 备份类型     | Access Key            | 备份地址                     | 最近更新时间           | 创建时间             |
|---------------|----------|-----------------------|--------------------------|------------------|------------------|
| mabing        | S3       | minio                 | http://10.6.178.185      | 2024-10-15 10:11 | 2024-10-15 10:11 |
| minio-default | 托管 MiniO | P1yfhGvGdLc0v/YFuf... | http://10.6.216.22:31... | 2024-07-29 14:51 | 2024-07-29 14:51 |

backup

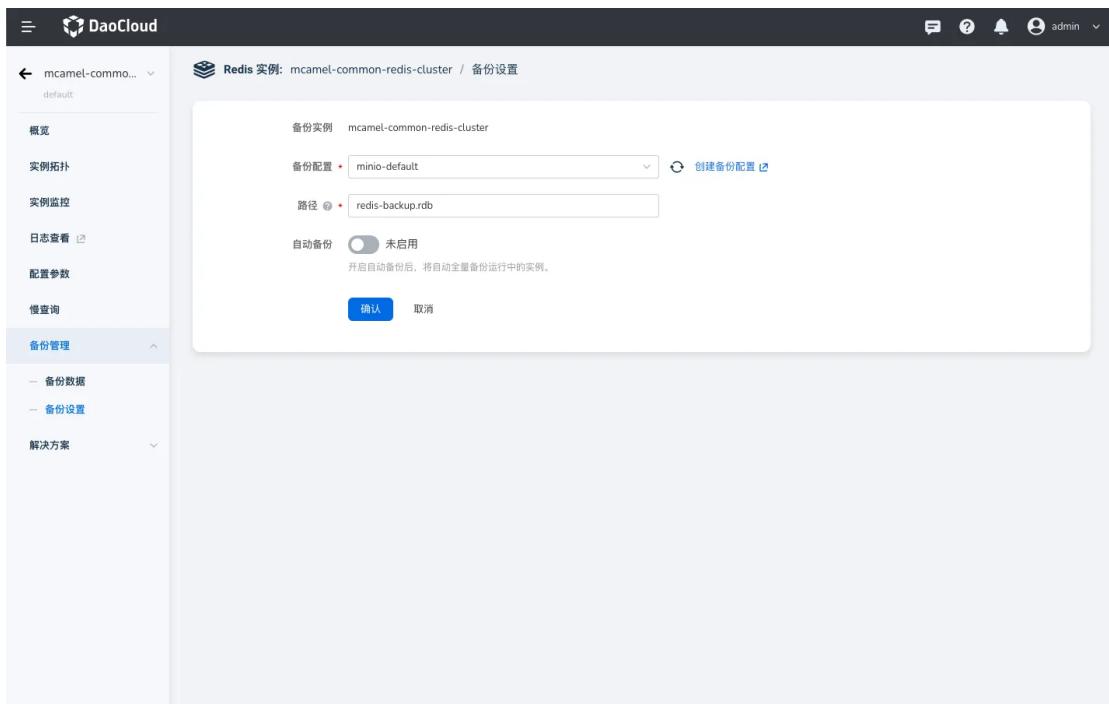
## 备份配置

1. 从实例列表中选择目标进行部分的 Redis 实例
2. 点击左侧菜单的 备份管理 -> 备份设置
3. 在当前页面选择备份存储的 S3 存储，并填写要存储的路径地址

### !!! note

Redis 备份数据通常通过 RDB 文件实现。在配置备份路径时，您只需指定存储文件夹名称和 .rdb 文件扩展名即可。

4. 如需每天或每周定时自定备份，可开启自动备份功能，选择备份的周期和时间。



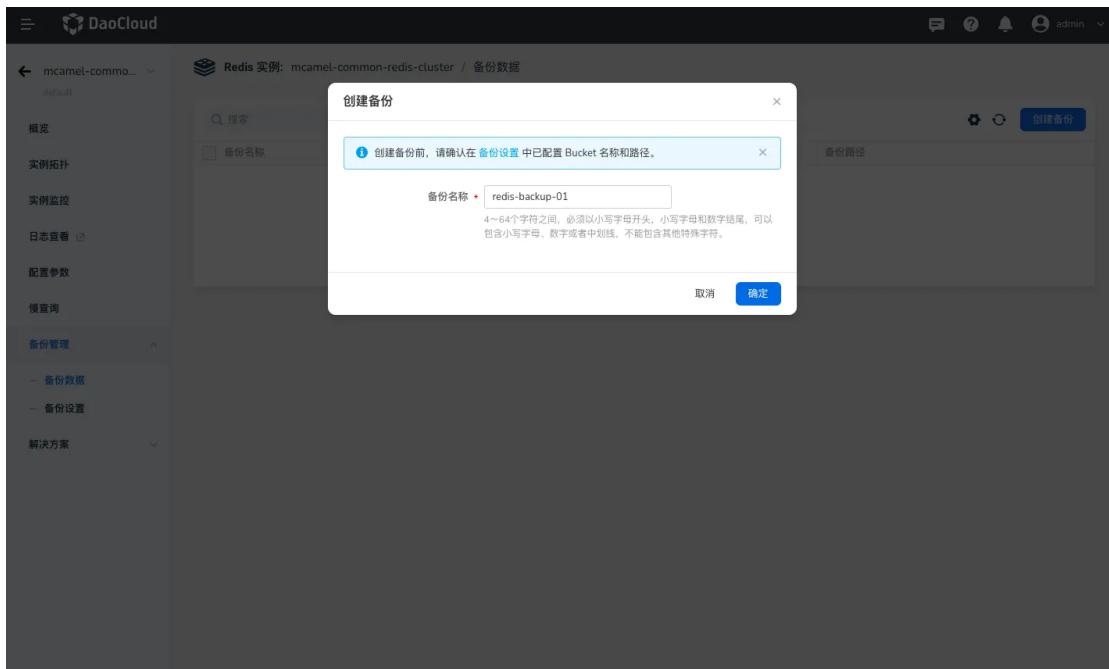
backup

## 备份数据

1. 从实例列表中选择目标进行部分的 Redis 实例

2. 点击左侧菜单的 备份管理 -> 备份数据

3. 点击列表右上的 创建备份 , 填写备份名称



backup

4. 点击**确定**后可在列表中查看备份状态、备份时间以及路径等信息。

The screenshot shows the DaoCloud Enterprise 5.0 web interface. On the left, there is a sidebar with various navigation options:概览 (Overview), 实例拓扑 (Instance Topology), 实例监控 (Instance Monitoring), 日志查看 (Log View), 配置参数 (Configuration Parameters), 慢查查询 (Slow Query), 备份管理 (Backup Management), and 解决方案 (Solution). The '备份管理' section is currently selected, and its sub-menu includes '备份数据' (Backup Data) and '备份设置' (Backup Settings). The main content area is titled 'Redis 实例: mccamel-common-redis-cluster / 备份数据'. It displays a table with one item: 'redis-backup-01' (备份名称), '备份成功' (Backup Status), '2024-10-29 16:12 - 2024-10-29 16:13' (Backup Start Time - Backup End Time), and 'redis-backup.rdb' (Backup Path). The table has columns for '备份名称' (Backup Name), '备份状态' (Backup Status), '备份开始时间-备份结束时间' (Backup Start Time - Backup End Time), and '备份路径' (Backup Path). There are also buttons for '搜索' (Search), '刷新' (Refresh), and '创建备份' (Create Backup).

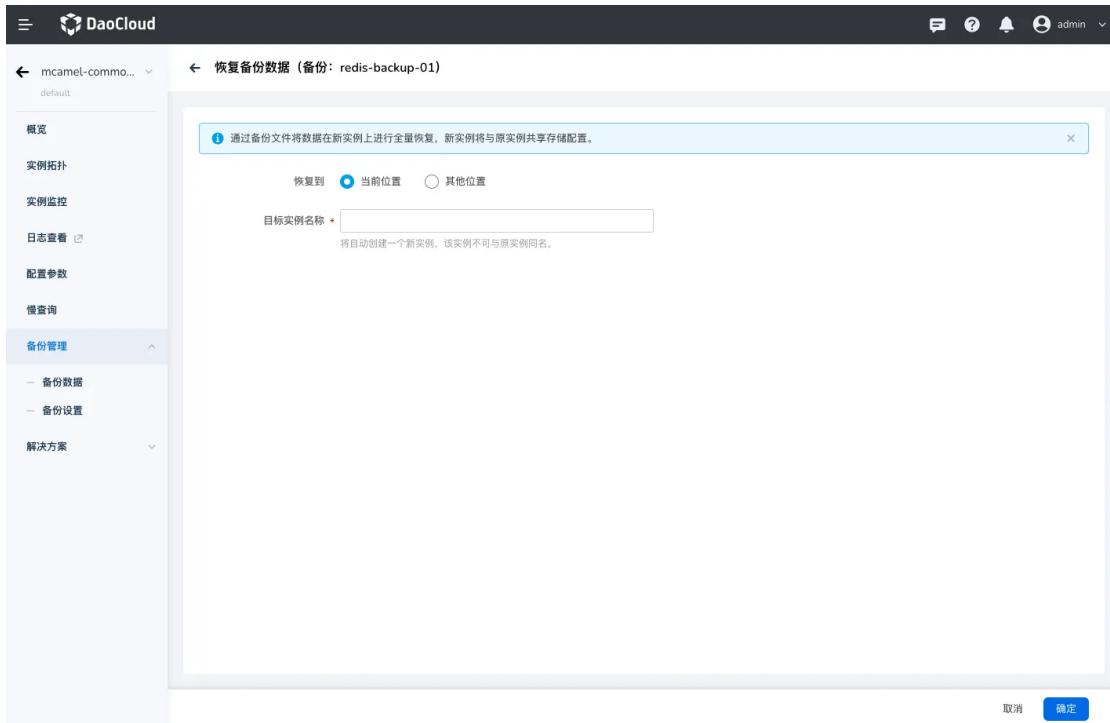
backup

## 恢复数据

1. 在备份数据列表中选择准备恢复的数据，**点击**备份数据列表最后一列的操作按钮，单

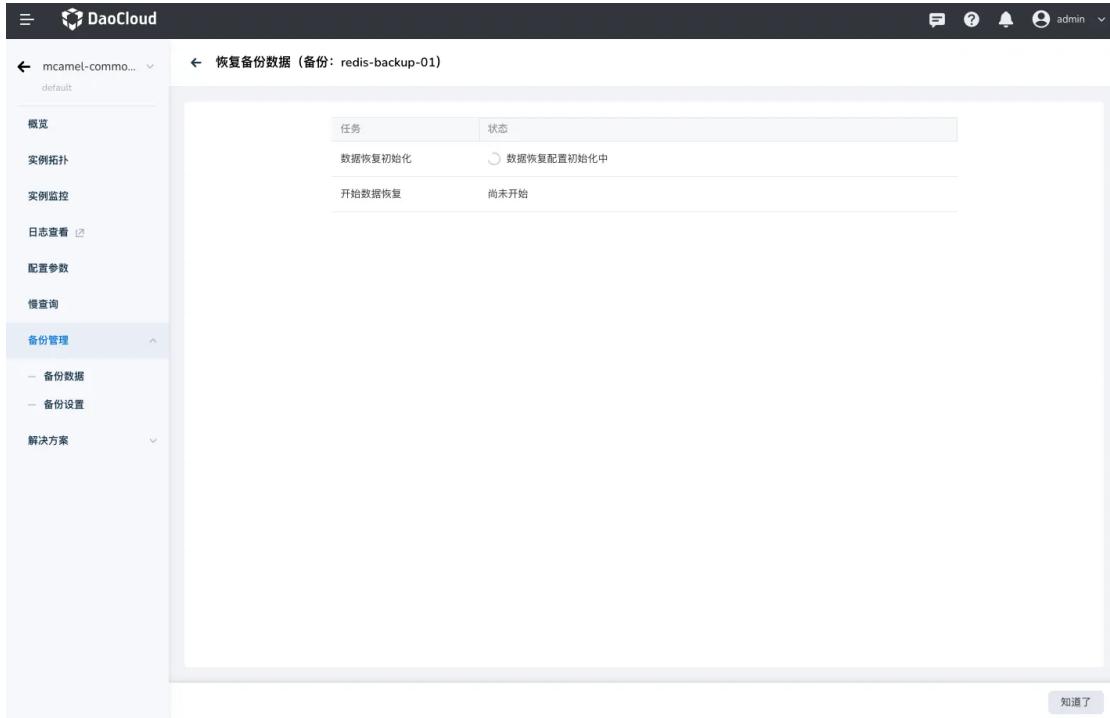
**击 恢复**

可选择恢复在当前实例所在集群和命名空间，或恢复到其他集群和命名空间。



alt text

2. 点击**确定**后，可查看数据恢复的状态。



backup

# 单集群跨机房高可用部署

## 场景需求

客户机房环境为单一 k8s 集群横跨 机房 A 、 机房 B ，期望可以部署一套 3 主 3 从集群模式 Redis ，实现跨机房高可用。希望当任一机房整体离线时， Redis 仍可以正常提供服务。

SVC  
SVC

## 解决方案

为了满足两机房高可用需要， Redis 副本需要采用以下部署方式：

- 3 个 leader 副本运行于集群节点： k8s-node-01 , k8s-node-02 , k8s-node-06
- 3 个 follower 副本运行于集群节点： k8s-node-03 , k8s-node-04 , k8s-node-05
- 确保每个集群节点仅运行一个 Redis 副本

本方案采用了工作负载的调度策略，通过具有权重的节点亲和性策略和工作负载反亲和策略达成以上部署目标。

!!! note

请保证各节点资源充足，避免因资源不足导致调度器无法完成正确调度。

## 1. 标签配置

### Redis 工作负载标签

本方案需要对 leader 副本与 follower 副本分别调度，因此用标签划分 redis 副本：

|              |                   |
|--------------|-------------------|
| Redis 副本     | 标签                |
| redis-leader | app: redis-leader |

Redis 副本 标签

redis-follower app: redis-follower

## 集群节点标签

为了把 Leader 与 follower 副本可控得分配在两个机房，需要在 6 个集群节点中划分两个拓扑域，用于分别调度 Leader 与 follower 副本。各集群节点的标签如下：

| 集群节点        | 标签          | 拓扑域 |
|-------------|-------------|-----|
| k8s-node-01 | az1: node01 | az1 |
| k8s-node-02 | az1: node02 | az1 |
| k8s-node-06 | az1: node03 | az1 |
| k8s-node-04 | az2: node01 | az2 |
| k8s-node-05 | az2: node02 | az2 |
| k8s-node-03 | az2: node03 | az2 |

## 2. 调度配置

redis-leader 和 redis-follower 副本需要调度至不同的拓扑域中，因此需分别配置亲和策略，配置如下：

### redis-leader

# redis-leader 在拓扑域 \_\_az1\_\_ 内的集群节点 (k8s-node-01, k8s-node-02, k8s-node-06) 执行工作负载反亲和，确保每个集群节点仅调度一个 leader 副本。

**affinity:**

podAntiAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

- labelSelector:

- matchExpressions:

- key: app

- operator: In

- values:

- redis-leader

topologyKey: az1

# redis-leader 的副本在拓扑域 \_\_az1\_\_ 内集群节点 (k8s-node-01, k8s-node-02, k8s-node-06) 亲和性调度

**nodeAffinity:**

preferredDuringSchedulingIgnoredDuringExecution:

```
- weight: 100
 preference:
 matchExpressions:
 - key: az1
 operator: In
 values:
 - node01
- weight: 90
 preference:
 matchExpressions:
 - key: az1
 operator: In
 values:
 - node02
- weight: 80
 preference:
 matchExpressions:
 - key: az1
 operator: In
 values:
 - node03
```

### redis-follower

```
redis-follower 的副本在拓扑域 __az2__ 内集群节点 (k8s-node-03, k8s-node-04, k8s-node-05) 工作负载反亲和
```

```
affinity:
 podAntiAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 - labelSelector:
 matchExpressions:
 - key: app
 operator: In
 values:
 - redis-follower
 topologyKey: az2
```

```
redis-follower 的副本在拓扑域 __az2__ 内集群节点 (k8s-node-03, k8s-node-04, k8s-node-05) 亲和性部署
```

```
nodeAffinity:
 preferredDuringSchedulingIgnoredDuringExecution:
 - weight: 100
 preference:
 matchExpressions:
 - key: az2
 operator: In
```

```

values:
 - node01
- weight: 90
preference:
 matchExpressions:
 - key: az2
 operator: In
 values:
 - node02
- weight: 80
preference:
 matchExpressions:
 - key: az2
 operator: In
 values:
 - node03

```

## 机房离线处理

### 机房 A 离线

机房 A 离线将导致两个 Redis-leader 副本离线，整个 Redis 将无法提供正常服务，如下图所示：

```

sync
sync

```

#### 解决办法

通过工具 redis-cli 进入 机房 B 中的任一 redis-follower 副本，手工转换为 leader 副本。

```

链接至一个 follower 节点
redis-cli -h <ip> -p <port>
密码验证，密码可在中间件模块的实例概览页查看
auth <password>
执行节点的角色转换
cluster failover takeover
查看节点角色信息，可见已改变
role

```

完成 机房 B 中副本角色转换后，集群可以恢复服务能力。当 机房 A 再次上线后，原 redis-leader 副本会以 follower 的角色接入 Redis 实例。

## 机房 B 离线

机房 B 离线将仅导致一个 redis-leader 副本离线，Redis 服务不会中断，无需人工干预，

如下图。

```
sync
sync
```

## 跨集群数据同步

Redis 作为开源的内存数据存储系统，具备出色的性能、高可用性和实时性能，可满足应用和服务的大量访问请求以及数据存储需求。然而，由于缺乏内置的数据安全保护功能，因此在实际应用中，需要采取一定的容灾技术来保障 Redis 数据的可靠性。

容灾技术主要通过数据复制和数据备份等手段实现，以提供冷备和热备两种备份方式。在网络、服务器或数据库发生故障导致数据丢失时，通过主备切换和数据恢复等方式，确保企业数据的安全性，确保业务的连续性。

Redis-Shake 是一个用于合并、同步和迁移 Redis 数据的工具，可以基于不同的 Redis 模式，提供相应的数据持续同步方案，本文将介绍不同模式间的数据同步与数据恢复配置方法。

- [集群模式 to 集群模式](#)
- [哨兵模式 to 哨兵模式](#)
- [集群模式 to 哨兵模式](#)

请参阅视频教程：[Redis 跨集群数据同步](#)。

# 集群模式 to 集群模式

[RedisShake](#) 支持集群模式实例间的数据同步与迁移能力，现以 3 主 3 从的集群模式为例，演示 Redis 跨集群同步配置方法。

假设实例 **redis-a** 与 **redis-b** 处于不同集群，二者均为 3 主 3 从的集群模式，现将 **redis-a** 作为主实例，**redis-b** 作为从实例搭建同步结构，提供以下灾备支持：

- 正常状态下，由 **redis-a** 对外提供服务，并持续同步数据到 **redis-b**
- 当主 **redis-a** 故障离线后，由 **redis-b** 对外提供服务
- 待 **redis-a** 恢复上线后，**redis-b** 向 **redis-a** 回写增量数据
- **redis-a** 数据恢复完成后，切换为初始状态，即由 **redis-a** 提供服务，并向 **redis-b** 持续数据同步

!!! note

数据同步与数据恢复需要通过不同的 RedisShake 组完成，因此在本例中实际要创建两组（Redis-shake-sync 和 Redis-shake-recovery）共 6 个 RedisShake 实例。

## 数据同步部署

图例：数据同步 **redis-a** -> **redis-b**

sync  
sync

## 为源端实例配置服务

如果源端实例处于 DCE 5.0 的集群中，可在 **数据服务 -> Redis -> 解决方案 -> 跨集群主从同步** 中开启方案，将自动完成服务配置工作。

SVC  
SVC

如果源端实例处于第三方集群上，则需要手工完成服务配置。为 Redis 实例的每个

Leader Pod 创建一个 **NodePort** 服务，用于 RedisShake 的数据同步访问。本例中需要为 **redis-a** 的 3 个 Leader Pod 分别创建服务。

下面以 Pod **redis-a-leader-0** 为例，为其创建服务：

1. 进入 **容器管理** -> **源端 Redis 实例所在集群** -> **有状态工作负载**，选择工作负载 **redis-a-leader**，为其创建一个服务，命名为 **redis-a-leader-svc-0**，访问类型为 **NodePort**，容器端口和服务端口均为 6379。

SVC  
SVC

2. 查看该服务。并确定工作负载选择器包含以下标签

app.kubernetes.io/component: redis  
app.kubernetes.io/managed-by: redis-operator  
app.kubernetes.io/name: redis-a  
app.kubernetes.io/part-of: redis-failover  
role: leader  
statefulset.kubernetes.io/pod-name: redis-a-leader-0 # (1)

1. 注意 **pod-name** 一定要选择正确的 leader pod 名称

SVC  
SVC

重复执行以上操作，为 **redis-a-leader-1**、**redis-a-leader-2** 分别创建服务。

## 部署 RedisShake

RedisShake 通常与数据传输的目标 Redis 实例运行于同一集群上。因此，本例中为了实现数据同步，需要在目标端部署 RedisShake。

注意，在集群模式下，RedisShake 要求与源端 Redis 实例的 Leader Pod 形成一对关系，因此这里需要部署 3 个独立的 RedisShake。以 **redis-a-leader-0** 为例，创建 Redis-shake-sync-0：

## 创建配置项

在 容器管理 -> 目标端 Redis 实例所在集群 -> 配置与存储 -> 配置项 中，为 RedisShake 实例创建配置项 **redis-sync-0**。导入文件 [sync.toml](#)，并注意需要修改以下内容：

```
conf
conf
• source.address : 源端 redis-a-leader-0 的 redis-a-leader-svc-0 服务地址 :
address = "10.233.109.145:31278"

• 源端实例的访问密码 : 可在 数据服务 实例的概览页获取该信息 :
password = "3wPxzWffdn" # keep empty if no authentication is required
SVC
SVC

• 目标端实例访问地址 , 此处需要填写目标端实例 redis-b 指向端口 6379 的 clusterIP
服务 , 这里使用 redis-b-leader 的地址 :
address = "10.233.43.13:6379"
```

您可以在 集群管理 -> 目标端所在集群 -> 工作负载 -> 访问方式 中查看此配置。

```
conf
conf
点击服务名称 , 进入服务详情 , 查看 ClusterIP 地址。
```

```
conf
conf
• 目标端实例的访问密码 , 可在 数据服务 模块下的 Redis 实例概览页获取该信息:
password = "3wPxzWffdn" # keep empty if no authentication is required

• 目标端类型需设置为 cluster :
[target]
type = "cluster" # "standalone" or "cluster"
```

## 创建 RedisShake

1. 打开 应用工作台 , 选择 向导 -> 基于容器镜像 , 创建一个应用

**redis-shake-sync-0 :**

```
sync
sync
```

2. 参考如下说明填写应用配置。

- 应用所属的集群和命名空间需与 Redis 实例一致；

- 镜像地址：

```
release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc
1e211fb9cb894c4dc85e94287395a32fc3dc
```

- 默认服务的访问类型为 NodePort，容器端口和服务端口设置为 6379。

```
sync
sync
```

- 在 高级设置 -> 生命周期 -> 启动命令 -> 运行参数 中填入：

```
/etc/sync/sync.toml
```

```
sync
sync
```

- 高级设置 -> 数据存储：添加配置项 **redis-sync-0**，路径必须设置为：

```
/etc/sync
```

- 高级设置 -> 数据存储：添加一个临时路径，容器路径必须为：

```
/data
```

```
sync
sync
```

3. 点击 确定，完成 RedisShake 创建。

重复执行以上操作，分别为其他两个 Leader Pod 创建 **redis-shake-sync-1**、

**redis-shake-sync-2**。

完成 RedisShake 的创建后，实际就已经开始 Redis 实例间的同步，此时可通过 [redis-cli](#) 工具验证同步情况。

## 数据恢复

图例：数据恢复 **redis-b** -> **redis-a**

```
recovery
recovery
```

当源端 **redis-a** 恢复上线后，首先需要从目标端 **redis-b** 恢复增量数据，因此需要在 **redis-a** 所在集群再次部署 3 个 RedisShake 实例，实现 **redis-b -> redis-a** 的数据回传。此处配置方法与数据同步过程类似，执行 **反方向** 配置部署即可。完成 RedisShake 创建后，即自动开始数据恢复。

#### !!! note

源端实例上线前，请先关闭当前正运行 **\_\_redis-b\_\_** 所在集群的 RedisShake 实例 **\_\_Redis-shake-sync-0\_\_**、  
**\_\_Redis-shake-sync-1\_\_**、**\_\_Redis-shake-sync-2\_\_**，避免发生错误的数据同步覆盖掉新增数据。

## 复原主从关系

如需复原初始的主从同步关系 **redis-a -> redis-b**，需在 **容器管理** 中停止用于数据恢复的 3 个 Redis-shake-recovery 实例，重新启动目标端集群中的 3 个 Redis-shake-sync 实例，即可重建初始的主从关系。

```
sync
sync
```

## 附录

```
```toml title="sync.toml" type = "sync"
[source] version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ... address =
"10.233.109.145: 6379" username = "" # keep empty if not using ACL password = "3wPxzWffdn"
# keep empty if no authentication is required tls = false elasticache_psyc = "" # using when
source is ElastiCache. ref: https://github.com/alibaba/RedisShake/issues/373
[target] type = "cluster" # "standalone" or "cluster" version = 6.0 # redis version, such as 2.8, 4.0,
5.0, 6.0, 6.2, 7.0, ... # When the target is a cluster, write the address of one of the nodes. #
RedisShake will obtain other nodes through the cluster nodes command. address =
"10.233.103.2: 6379" username = "" # keep empty if not using ACL password = "Aa123456" #
keep empty if no authentication is required tls = false
[advanced] dir = "data"
```

**runtime.GOMAXPROCS, 0 means use
runtime.NumCPU() cpu cores**

ncpu = 4

pprof port, 0 means disable

pprof_port = 0

metric port, 0 means disable

metrics_port = 0

log

log_file = "redis-shake.log" log_level = "info" # debug, info or warn log_interval = 5 # in seconds

RedisShake gets key and value from rdb file, and uses RESTORE command to

create the key in target redis. Redis RESTORE will return a “Target key name

is busy” error when key already exists. You can use this configuration item

to change the default behavior of restore:

panic: RedisShake will stop when meet “Target key name is busy” error.

rewrite: RedisShake will replace the key with new value.

ignore: RedisShake will skip restore the key when meet “Target key name is busy” error.

```
rdb_restore_command_behavior = "rewrite" # panic, rewrite or skip
```

pipeline

pipeline_count_limit = 1024

Client query buffers accumulate new commands. They are limited to a fixed amount by default. This amount is normally 1gb.

target_redis_client_max_querybuf_len = 1024_000_000

In the Redis protocol, bulk requests, that are, elements representing single strings, are normally limited to 512 mb.

target_redis_proto_max_bulk_len = 512_000_000

哨兵模式 to 哨兵模式

[RedisShake](<https://tair.opensource.github.io/RedisShake/>) 支持哨兵模式实例间的数据同步与迁移能力，现以 3 副本哨兵模式 Redis 为例，演示跨集群同步配置方法。

假设 __实例 redis-a__ 与 __实例 redis-b__ 处于不同集群，二者均为 3 副本哨兵模式，现将 __实例 redis-a__ 作为主实例， __实例 redis-b__ 作为从实例搭建同步结构，提供以下灾备支持：

- 正常状态下，由 __实例 redis-a__ 对外提供服务，并持续同步数据 __实例 redis-a__ -> __实例 redis-b__；
- 当主 __实例 redis-a__ 故障离线后，由 __实例 redis-b__ 对外提供服务；
- 待 __实例 redis-a__ 恢复上线后， __实例 redis-b__ -> __实例 redis-a__ 回写增量数据

；
 - 实例 redis-a__ 数据恢复完成后，切换为初始状态，即由 实例 redis-a__ 提供服务，并向 实例 redis-b__ 持续数据同步。

!!! note

数据同步与数据恢复需要通过不同的 RedisShake 组完成，因此在本例中需要分别创建 Redis-shake-sync__、Redis-shake-recovery__ 两个 RedisShake 实例。

数据同步部署

图例：数据同步 实例 redis-a__ → 实例 redis-b__

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync12.png)

为源端实例配置服务

如果源端实例处于 DCE 5.0 的集群中，可在 数据服务__ → Redis__ → 解决方案__ → 跨集群主从同步__ 中开启方案，将自动完成服务配置工作。

![svc](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync17.png)

如果源端实例处于第三方集群上，则需要手工完成服务配置，配置方法如下文：

为 Redis 实例创建一个 Nodeport__ 服务，用于 RedisShake 的数据同步访问。本例中需要为 实例 redis-a__ 创建 1 个服务，以下为创建过程：

1. 进入 容器管理__ → 源端实例所在集群__ → 有状态工作负载__：选择工作负载 redis-a__，创建一个 Nodeport__ 服务，容器端口和服务端口均为 6379

![svc](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync03.png)

2. 查看该服务。并确定工作负载选择器包含以下标签

```
```yaml
app.kubernetes.io/component: redis
app.kubernetes.io/managed-by: redis-operator
app.kubernetes.io/name: redis-a
app.kubernetes.io/part-of: redis-failover
redis-failover: master
redisfailovers.databases.spotahome.com/name: redis-a
```

```

```
```

```

![svc]([https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync14.png\)](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync14.png)

#### ### RedisShake 部署

RedisShake 通常与数据传输的目标 Redis 实例运行于同一集群上，因此，本例中为了实现数据同步，需要在 实例 redis-b 所在集群部署 RedisShake，配置方式如下。

#### #### 创建配置项

在 容器管理 -> 目标端实例所在集群 -> 配置与存储 -> 配置项 为 RedisShake 实例创建配置项 redis-sync。导入文件 sync.toml（文件内容见 附录），并注意需要修改以下内容：

![conf](<https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync15.jpg>)

- source.address：上一步骤创建的源端 redis-a 的服务地址：

```
```toml
address = "10.233.109.145:31278"
```

```

- 源端实例的访问密码：可在 Redis 实例的概览页获取该信息：

```
```toml
password = "3wPxzWffdn" # keep empty if no authentication is required
```
![conf](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync16.png\)
```

- 目标端实例访问地址，该地址可以采用新建的 ClusterIP 服务地址，也可以采用实例 redis-b 的默认 Headless 服务 rfr-redis-b 的地址，本例中采用 Headless 服务地址：

```
```toml
address = "rfr-redis-b:6379"
```

```

该服务信息可在 集群管理 -> 目标端所在集群 -> 工作负载 -> 访问方式 中查看。类似下图所示页面

![conf](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync22.png)

- 目标端实例的访问密码，可在 数据服务 模块下的 Redis 实例概览页获取该信息：

```
```toml
password = "3wPxzWffdn" # keep empty if no authentication is required
```

```

类似下图位置：

![svc](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync06.png)

- 目标端类型需设置为 standalone：

```
```toml
[target]
type = "standalone" # "standalone" or "cluster"
```

```

#### #### 创建 RedisShake

1. 打开 应用工作台，选择 向导 -> 基于容器镜像，创建一个应用 Redis-shake-sync：

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync07.png)

2. 参考如下说明填写应用配置。

- 应用所属集群、命名空间需与 Redis 实例一致；
- 镜像地址：

```
```yaml
release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc1e21
1fb9cb894c4dc85e94287395a32fc3dc
```

```

- 默认服务的访问类型为 Nodeport，容器端口和服务端口设置为 6379。

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync08.png)

- 高级设置 -> 生命周期 -> 启动命令 -> 运行参数 填入:

```
```yaml
/etc-sync/sync.toml
```

```

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync09.png)

- 高级设置 -> 数据存储 : 添加配置项 redis-sync , 路径必须设置为:

```
```yaml
/etc-sync
```

```

- 高级设置 -> 数据存储 : 添加一个临时路径, 容器路径必须为:

```
```yaml
/data
```

```

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync20.png)

3. 点击 确定 , 完成 RedisShake 创建。

完成 RedisShake 的创建后, 实际就已经开始 Redis 实例间的同步, 此时可通过 redis-cli 工具验证同步, 这里就不做赘述。

## 数据恢复

图例: 数据恢复 实例 redis-b -> 实例 redis-a

![recovery](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync13.png)

当源端 实例 redis-a 恢复上线后, 首先需要从目标端 实例 redis-b 恢复增量数据, 因此需要在 实例 redis-a 再次部署 1 个 RedisShake 实例, 实现 实例 redis-b -> 实例 redis-a 的数据回传, 此处配置方法与数据同步过程类似, 执行 **\*\*反方向\*\*** 配置部署即可, 完成 RedisShake 创建后, 即自动开始数据恢复。

**!!! note**

源端实例上线前, 请先关闭用于正常同步的 Redis-shake-sync , 避免发生错误的数据

同步覆盖掉新增数据。

## ## 复原主从关系

如需复原初始的主从同步关系 `实例 redis-a` → `实例 redis-b`，需在 `容器管理` 中停止 `实例 redis-a` 所在集群中正在运行的 `Redis-shake-recovery` 实例，重新启动目标端集群中的 `Redis-shake-sync` 实例，即可重建初始主从关系。

![sync]([https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync11.png\)](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/sync11.png)

## ## 附录

```
```toml title="sync.toml"
type = "sync"

[source]
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
address = "10.233.109.145: 6379"
username = "" # keep empty if not using ACL
password = "3wPxzWffdn" # keep empty if no authentication is required
tls = false
elasticache_psync = "" # using when source is ElastiCache. ref: https://github.com/alibaba/RedisShake/issues/373

[target]
type = "standalone" # "standalone" or "cluster"
version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ...
# When the target is a cluster, write the address of one of the nodes.
# RedisShake will obtain other nodes through the cluster nodes command.
address = "10.233.103.2: 6379"
username = "" # keep empty if not using ACL
password = "Aa123456" # keep empty if no authentication is required
tls = false

[advanced]
dir = "data"

# runtime.GOMAXPROCS, 0 means use runtime.NumCPU() cpu cores
ncpu = 4

# pprof port, 0 means disable
pprof_port = 0
```

```

# metric port, 0 means disable
metrics_port = 0

# log
log_file = "redis-shake.log"
log_level = "info" # debug, info or warn
log_interval = 5 # in seconds

# redis-shake gets key and value from rdb file, and uses RESTORE command to
# create the key in target redis. Redis RESTORE will return a "Target key name
# is busy" error when key already exists. You can use this configuration item
# to change the default behavior of restore:
# panic: redis-shake will stop when meet "Target key name is busy" error.
# rewrite: redis-shake will replace the key with new value.
# ignore: redis-shake will skip restore the key when meet "Target key name is busy" error.
rdb_restore_command_behavior = "rewrite" # panic, rewrite or skip

# pipeline
pipeline_count_limit = 1024

# Client query buffers accumulate new commands. They are limited to a fixed
# amount by default. This amount is normally 1gb.
target_redis_client_max_querybuf_len = 1024_000_000

# In the Redis protocol, bulk requests, that are, elements representing single
# strings, are normally limited to 512 mb.
target_redis_proto_max_bulk_len = 512_000_000

```

集群模式 to 哨兵模式

RedisShake 支持不同部署模式实例间的数据同步与迁移能力，现以 3 主 3 从集群模式实例与 3 副本哨兵模式实例的场景为例，演示不同模式之间的同步配置方法。

假设 Redis 实例 `redis-a` 为 3 主 3 从集群模式，实例 `redis-b` 为 3 副本哨兵模式，两实例处于不同集群。现将 实例 `redis-a` 作为主实例，实例 `redis-b` 作为从实例搭建同步结构，提供以下灾备支持：

- 正常状态下，由 实例 `redis-a` 对外提供服务，并持续同步数据 实例 `redis-a` -> 实例

```
redis-b ;
```

- 当主 实例 redis-a 故障离线后，由 实例 redis-b 对外提供服务；
- 待 实例 redis-a 恢复上线后，实例 redis-b -> 实例 redis-a 回写增量数据；
- 实例 redis-a 数据恢复完成后，切换为初始状态，即由 实例 redis-a 提供服务，并向 实例 redis-b 持续数据同步。

!!! note

数据传输的方式与传输源端实例有关，在 数据同步 中源端 实例 redis-a 为集群模式，需要为 3 个 leader 副本分别部署一个 RedisShake (Redis-shake-sync)；在 数据恢复 中源端 实例 redis-b 为哨兵模式，则仅需部署一个 RedisShake (Redis-shake-recovery)。

数据同步部署

图例：数据同步 实例 redis-a -> 实例 redis-b

```
sync  
sync
```

为 实例 redis-a 配置服务

为 Redis 实例的每一个 Leader Pod 创建一个 **Nodeport** 服务，用于 RedisShake 的数据同步访问。本例中需要为 实例 redis-a 的 3 个 Leader Pod 分别创建服务，下面以 Pod **redis-a-leader-0** 为例，为其创建服务：

1. 进入 **容器管理** -> **源端实例所在集群** -> **有状态工作负载**：选择工作负载 **redis-a-leader**，为其创建一个服务，命名为 **redis-a-leader-svc-0**，访问类型为 **Nodeport**，容器端口和服务端口均为 6379。

```
SVC  
SVC
```

2. 查看该服务。并确定工作负载选择器包含以下标签

```
app.kubernetes.io/component: redis  
app.kubernetes.io/managed-by: redis-operator
```

```

app.kubernetes.io/name: redis-a
role: leader
# 注意 pod-name 一定要选择正确的 leader pod 名称
statefulset.kubernetes.io/pod-name: redis-a-leader-0
SVC
SVC

```

重复执行以上操作，为 **redis-a-leader-1** **redis-a-leader-2** 分别创建服务。

为 实例 **redis-b** 配置服务

参考 图例：数据同步 实例 **redis-a** -> 实例 **redis-b** 可见， 实例 **redis-b** 与 RedisShake 处于同一集群， 因此为 实例 **redis-b** 创建一个 ClusterIP 服务即可,端口指向 6379，如下图所示：

```

SVC
SVC
SVC
SVC

```

RedisShake 部署

RedisShake 通常与数据传输的目标 Redis 实例运行于同一集群及命名空间上，因此，本例中为了实现数据同步，需要在 实例 **redis-b** 所在集群部署 RedisShake，配置方式如下。

注意，在集群模式下，RedisShake 要求与源端 Redis 实例的 Leader Pod 形成一对关系（请参考图例：数据同步 实例 **redis-a** -> 实例 **redis-b**），因此这里需要部署 3 个独立的 RedisShake。以 **redis-a-leader-0** 为例，创建 **Redis-shake-sync-0**：

1. 创建配置项

在 容器管理 -> 目标端实例所在集群 -> 配置与存储 -> 配置项 为 RedisShake 实例创建配置项 **redis-sync-0**。导入文件 **sync.toml**（文件内容见附录），并注意需要修改以下内

容：

- conf
- conf
 - source.address : 源端 实例 **redis-a** 的 **redis-a-leader-svc-0** 服务地址 :
 - address = "10.233.109.145:32283"**
 - 源端实例的访问密码 : 可在 **数据服务** 实例的概览页获取该信息 :
 - password = "3wPxzWffdn" # keep empty if no authentication is required**
 - SVC
 - SVC
 - 目标端实例访问地址 , 采用上一步为 实例 **redis-b** 创建的 ClusterIP 服务 (端口 6379)

作为目标端的访问地址 :

```
# 注意: 由于 headless 服务可直接连接工作负载, 因此可以通过该服务访问工作负载默
认端口 6379。
address = "10.233.16.241:6379"
```

- 目标端实例的访问密码 , 可在 **数据服务** 模块下的 Redis 实例概览页获取该信息:
- password = "3wPxzasd" # keep empty if no authentication is required**
- 目标端类型需设置为 **standalone** :

[target]
type = "standalone" # "standalone" or "cluster"

2. 创建 RedisShake

1. 打开 应用工作台 , 选择 向导 -> 基于容器镜像 , 创建一个应用

Redis-shake-sync-0 :

```
sync
sync
```

2. 参考如下说明填写应用配置。

- 应用所属集群、命名空间需与 Redis 实例一致 ;
- 镜像地址 :

```
release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc
1e211fb9cb894c4dc85e94287395a32fc3dc
```

- 默认服务的访问类型为 Nodeport，容器端口和服务端口设置为 6379。

```
sync
sync
```

- 高级设置 -> 生命周期 -> 启动命令 -> 运行参数 填入：

```
/etc/sync/sync.toml
sync
sync
```

- 高级设置 -> 数据存储：添加配置项 redis-sync-0，路径必须设置为：

```
/etc/sync
```

- 高级设置 -> 数据存储：添加一个临时路径，读写权限为 读写，容器路径必须为：

```
/data
sync
sync
```

3. 点击 **确定**，完成 RedisShake 创建。

重复执行以上操作，分别为其他两个 Leader Pod 创建 **Redis-shake-sync-1**，**Redis-shake-sync-2**。

完成 RedisShake 的创建后，实际就已经开始 Redis 实例间的同步，此时可通过 **redis-cli** 工具验证同步，这里就不做赘述。

数据恢复

图例：数据恢复 实例 **redis-b** -> 实例 **redis-a**

```
recovery
recovery
```

当源端实例 实例 **redis-a** 发生故障离线后，将由目标端 实例 **redis-b** 提供服务，该过程必然产生新增数据。 实例 **redis-a** 恢复上线后，首先需要从 实例 **redis-b** 恢复增量数据，此时 实例 **redis-a** 与 实例 **redis-b** 角色互换， 实例 **redis-b** 作为数据源向 实例 **redis-a** 同步数据。

参考 图例：数据恢复实例 **redis-b** -> 实例 **redis-a**，因 实例 **redis-a** 为集群模式，需要在其所在集群部署 1 个 RedisShake 实例，实现 实例 **redis-b** -> 实例 **redis-a** 的数据回传，配置方式如下。

!!! note

实例 **redis-a** 上线前，请先关闭用于实例 **redis-a** -> 实例 **redis-b** 数据同步的 3 个 RedisShake 实例(**_redis-shake-sync-0_**、**_redis-shake-sync-1_**、**_redis-shake-sync-2_**)，避免发生错误的数据同步覆盖掉新增数据。

为 实例 **redis-b** 配置服务

恢复流程中 实例 **redis-b** 为数据源，因此需要为该实例创建一个 Nodeport 服务，用于 RedisShake 的跨集群访问。

此时的数据源 实例 **redis-b** 为哨兵模式，因此仅需要创建 1 个服务，以下为创建过程：

1. 进入 容器管理 -> 源端实例所在集群 -> 有状态工作负载：选择工作负载 **redis-b**，

创建一个 Nodeport 服务 **redis-b-recovery-svc**，容器端口和服务端口均为 6379

SVC
SVC

2. 查看该服务。并确定工作负载选择器包含以下标签

```
app.kubernetes.io/component: redis
app.kubernetes.io/managed-by: redis-operator
app.kubernetes.io/name: redis-b
app.kubernetes.io/part-of: redis-failover
redis-failover: master
redisfailovers.databases.spotahome.com/name: redis-b
                                         SVC
                                         SVC
```

RedisShake 部署

RedisShake 通常与数据传输的目标 Redis 实例运行于同一集群上，因此，本例中为了实现数据同步，需要在目标端部署 RedisShake，配置方式如下。

1. 创建配置项

在 容器管理 -> 目标端实例所在集群 -> 配置与存储 -> 配置项 为 RedisShake 实例创建配置项 **redis-sync**。导入文件 **sync.toml**（文件内容见 [附录](#)），并注意需要修改以下内容：

```
conf
conf
```

- source.address：此时的源端为 实例 **redis-b**，填写上一步骤为该实例创建的服务地址：

```
address = "10.233.109.145:32283"
```

- 源端实例的访问密码：可在 实例 **redis-b** 的概览页获取该信息：

```
password = "3wPxzWffdn" # keep empty if no authentication is required
conf
conf
```

- 目标端实例访问地址，此时的目标端为 实例 **redis-a**，这里可以采用系统默认创建的服务 **redis-a-leader**（端口：6379），无需创建：

```
address = "172.30.120.202:6379"
```

该配置可在 集群管理 -> **redis-a** 所在集群 -> 工作负载 -> 访问方式 中查看。如下图所示：

```
conf
conf
```

点击服务名称，进入服务详情，可见 ClusterIP 地址：

- ```
conf
conf
```
- 目标端实例的访问密码，可在 实例 **redis-a** 的概览页获取该信息：
- ```
password = "3wPxzWffss" # keep empty if no authentication is required
```
- 此时数据接收端为 实例 **redis-b**，因此目标端类型需设置为 **cluster**：
- [target]**
- ```
type = "cluster" # "standalone" or "cluster"
```

## 2. 创建 RedisShake

1. 打开 应用工作台 , 选择 向导 -> 基于容器镜像 , 创建一个应用

**Redis-shake-recovery** :

```
sync
sync
```

2. 参考如下说明填写应用配置。

- 应用所属集群、命名空间需与 Redis 实例一致 ;

- 镜像地址 :

`release.daocloud.io/ndx-product/redis-shake@sha256:46652d7d8893fa4508c3c6725afc  
1e211fb9cb894c4dc85e94287395a32fc3dc`

- 默认服务的访问类型为 Nodeport , 容器端口和服务端口设置为 6379。

```
sync
sync
```

- 高级设置 -> 生命周期 -> 启动命令 -> 运行参数 填入 :

`/etc/sync/sync.toml`  
sync  
sync

- 高级设置 -> 数据存储 : 添加配置项 `redis-sync` , 路径必须设置为 :

`/etc/sync`

- 高级设置 -> 数据存储 : 添加一个临时路径 , 容器路径必须为 :

`/data`  
sync  
sync

3. 点击 确定 , 完成 RedisShake 创建。

完成 RedisShake 的创建后 , 实际就已经开始 Redis 实例间的同步 , 此时可通过 `redis-cli` 工具验证同步 , 这里就不做赘述。

## 复原主从关系

如需复原初始的主从同步关系 实例 redis-a -> 实例 redis-b , 需在 容器管理 中停止当前运行在 实例 redis-a 所在集群的 redis-shake-recovery 实例 , 重新启动目标端实例所在集群中的 3 个 redis-shake-sync 实例 , 即可重建初始主从关系。

```
sync
sync
```

## 附录

```
```toml title="sync.toml" type = "sync"
[source] version = 6.0 # redis version, such as 2.8, 4.0, 5.0, 6.0, 6.2, 7.0, ... address =
"10.233.109.145: 6379" username = "" # keep empty if not using ACL password = "3wPxzWffdn"
# keep empty if no authentication is required tls = false elasticache_psync = "" # using when
source is ElastiCache. ref: https://github.com/alibaba/RedisShake/issues/373
[target] type = "cluster" # "standalone" or "cluster" version = 6.0 # redis version, such as 2.8, 4.0,
5.0, 6.0, 6.2, 7.0, ... # When the target is a cluster, write the address of one of the nodes. #
RedisShake will obtain other nodes through the cluster nodes command. address =
"10.233.103.2: 6379" username = "" # keep empty if not using ACL password = "Aa123456" #
keep empty if no authentication is required tls = false
[advanced] dir = "data"
```

runtime.GOMAXPROCS, 0 means use runtime.NumCPU() cpu cores

```
ncpu = 4
```

pprof port, 0 means disable

```
pprof_port = 0
```

metric port, 0 means disable

```
metrics_port = 0
```

log

```
log_file = "redis-shake.log" log_level = "info" # debug, info or warn log_interval = 5 # in seconds
```

**redis-shake gets key and value from rdb file,
and uses RESTORE command to**

**create the key in target redis. Redis RESTORE
will return a “Target key name**

**is busy” error when key already exists. You
can use this configuration item**

to change the default behavior of restore:

**panic: redis-shake will stop when meet
“Target key name is busy” error.**

**rewrite: redis-shake will replace the key with
new value.**

**ignore: redis-shake will skip restore the key
when meet “Target key name is busy” error.**

```
rdb_restore_command_behavior = "rewrite" # panic, rewrite or skip
```

pipeline

```
pipeline_count_limit = 1024
```

Client query buffers accumulate new commands. They are limited to a fixed amount by default. This amount is normally 1gb.

```
target_redis_client_max_querybuf_len = 1024_000_000
```

In the Redis protocol, bulk requests, that are, elements representing single

strings, are normally limited to 512 mb.

```
target_redis_proto_max_bulk_len = 512_000_000
```

```
# 基于 Spiderpool 的外部访问
```

```
## 配置目标
```

由于 Redis 本身并未提供集群外部访问的能力，因此需要通过其他网络功能，实现该访问需求。

本例中采用了 calico/cilium + macvlan standalone + spiderpool 的支持方式。

```
## 前提
```

DCE 5.0 集群内已部署 __multus-underlay__ 和 __spiderpool__。

ool09.png)

环境准备

- 确认 macvlan 部署情况。执行命令，查看部署状态，可见类似如下图中的返回信息

```
```shell
kubectl get network-attachment-definitions -A
```

```

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool01.png)

存在 `macvlan-vlan0(standalone)` 和 `macvlan-overlay-vlan0(overlay)` 则表示集群内已部署 macvlan。

!!! note

Redis 仅支持通过 `macvlan standalone` 模式实现外部访问

- 创建子网及 IP 池，具体操作可参见：[创建子网及 IP 池](https://docs.daocloud.io/network/modules/spiderpool/createpool.html)

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool02.png)

!!! note

Redis 仅可以使用现有子网及 IP 池，请务必先执行手工创建操作。

Redis 实例配置

集群模式

- 修改 redis 实例的 `CR` (`rediscluster`)，在 `metadata` 字段下添加以下内容：

```
```yaml
annotations:
 v1.multus-cni.io/default-network: kube-system/macvlan-vlan0
 ipam.spidernet.io/ippools: '[{"interface":"eth0","ipv4":["ippool-redis"]}]'
```

```

- 更新 `CR` 后，查看实例节点信息，可见类似下图的 IP 地址变化，则表示配置成功：

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool03.png)

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool04.png)

3. 完成配置后可从集群外部访问节点，能成功访问。

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool05.png)

哨兵模式

1. 更新 Redis 实例 `_CR_` (`redisfailover`)，分别在 `_spec.redis_` 和 `_spec.sentinel_` 字段添加以下内容：

```
```yaml
podAnnotations:
 v1.multus-cni.io/default-network: kube-system/macvlan-vlan0
 ipam.spidernet.io/ippools: '[{"interface":"eth0","ipv4":["ippool-redis"]}]'
```

```

!!! note

cilium: 需要为 `_redis-operator_` 的 `_deployment_` 添加 `_annotations_`，
字段位置为 `_spec.template.metadata.annotations_`

calico: 无需更新 `_redis-operator_`

2. 更新 `_CR_` 后，查看实例节点信息，可见类似下图的 IP 地址变化，则表示配置成功：

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool06.png)

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool07.png)

3. 完成配置后可从集群外部访问节点验证配置有效性，能成功访问。

![sync](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/redis/images/spiderpool08.png)

!!! note "注意事项"

Reids 哨兵模式中的工作负载 `updateStrategy` 为 `OnDelete` 时，更新 CR 之后不会立即删除旧版本的 Pod，需要手动去重启 Pod，重启 Pod 即可生效。

下图为更新 CR 后未更新的数据截图：

![cr-pod](docs/zh/docs/middleware/redis/images/ippool-pod-not-restart.png)

什么是 Elasticsearch

Elasticsearch（下文简称 Elastic）是目前全文搜索引擎的首选。它可以快速地存储、搜索和分析海量数据。Elastic 的底层是开源库 Lucene，但是 Lucene 无法直接用，必须自己写代码去调用它的接口。Elastic 是 Lucene 的封装，提供了 REST API 的操作接口，开箱即用。

DCE 5.0 内置的搜索服务基于 Elasticsearch，能够提供分布式搜索服务，为用户提供结构化、非结构化文本以及基于 AI 向量的多条件检索、统计、报表。完全兼容 Elasticsearch 原生接口。它可以帮助网站和 APP 搭建搜索框，提升用户的搜索体验；也可以用于搭建日志分析平台，助力企业实现数据驱动运维，数据驱动运营；它的向量检索能力可以帮助客户快速构建基于 AI 的图搜、推荐、语义搜索、人脸识别等丰富的应用。

![欢迎界面](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/es01.png)

Elasticsearch 的工作管理

原始数据会从多个来源（包括日志、系统指标和网络应用程序）输入到 Elasticsearch 中。数据采集旨在 Elasticsearch 中进行索引之前解析、标准化并充实这些原始数据的过程。这些数据在 Elasticsearch 中完成索引之后，用户便可针对数据运行复杂的查询，并使用聚合来检索自身数据的复杂汇总。在 Kibana 中，用户可以基于自己的数据创建强大的可视化。

Kibana 是什么？

Kibana 是一款适用于 Elasticsearch 的数据可视化和管理工具，可以提供实时的直方图、线形图、饼状图和地图。Kibana 同时还包括诸如 Canvas 和 Elastic Maps 等高级应用程序；其中 Canvas 允许用户基于自身数据创建定制的动态信息图表，而 Elastic Maps 则可用来对地理空间数据进行可视化。

[创建 Elasticsearch 实例](./user-guide/create.md){ .md-button .md-button--primary }

功能说明

Elasticsearch 以 JSON 文档的形式存储数据。

每个文档都会在一组键（字段或属性的名称）与它们对应的值（字符串、数字、布尔值、日期、

数值组、地理位置或其他类型的数据) 之间建立联系。

Elasticsearch 使用的是一种名为倒排索引的数据结构，这一结构的设计可以允许十分快速地进行全文本搜索。

倒排索引会列出在所有文档中出现的每个特有词汇，并且可以找到包含每个词汇的全部文档。

在索引过程中，Elasticsearch 会存储文档并构建倒排索引，这样用户便可以近实时地对文档数据进行搜索。

索引过程是在索引 API 中启动的，通过此 API 您既可向特定索引中添加 JSON 文档，也可更改特定索引中的 JSON 文档。

Elasticsearch 支持的通用功能特性如下：

| 分类 | 特性 | 说明 |
|---|----|-----|
| ----- ----- ----- | | |
| 分布式集群 集群部署、集群监控
群、节点、索引的运行状况 | | 监控集 |
| 搜索管理 索引配置管理、结构定义、索引重建
管理平台提供配置功能 | | 搜索 |
| 全文搜索 搜索功能、排序功能、统计分析功能
过 RESTful API 方式提供 | | 通 |
| 数据采集 ElasticSearch 数据导入 API、Maxcompute 数据导入工具、全量/增量采集方式
丰富的原生数据采集接口，集成 Maxcompute 数据导入工具 | | |
| 服务鉴权 服务级别的用户鉴权机制
用户鉴权设置 | | 统一的 |

在 DCE 5.0 中部署 Elasticsearch 后，还将支持以下特性：

- 支持 Elasticsearch 专有节点、热数据节点、冷数据节点、数据节点角色部署
- 集成 Kibana
- 基于 elasticsearch-exporter 暴露指标
- 基于 Grafana Operator 集成 Elasticsearch Dashboard，展示监控数据
- 使用 ServiceMonitor 对接 Prometheus 抓取指标
- 基于[工作空间 Workspace](../../../../ghippo/user-guide/workspace/workspace.md) 多租户化管理

产品优势

Elasticsearch 很快

由于 Elasticsearch 是在 Lucene 基础上构建而成的，所以在全文本搜索方面表现十分出色。Elasticsearch 同时还是一个近实时的搜索平台，这意味着从文档索引操作到文档变为可搜索状

态之间的延时很短，一般只有一秒。

因此，Elasticsearch 非常适用于对时间有严苛要求的用例，例如安全分析和基础设施监测。

Elasticsearch 具有分布式的本质特征

Elasticsearch 中存储的文档分布在不同的容器中，这些容器称为分片，可以进行复制以提供数据冗余副本，以防发生硬件故障。

Elasticsearch 的分布式特性使得它可以扩展至数百台（甚至数千台）服务器，并处理 PB 量级的数据。

Elasticsearch 包含一系列广泛的功能

除了速度、可扩展性和弹性等优势以外，Elasticsearch 还有大量强大的内置功能（例如数据汇总和索引生命周期管理），可以方便用户更加高效地存储和搜索数据。

Elastic Stack 简化了数据采集、可视化和报告过程

通过与 Beats 和 Logstash 进行集成，用户能够在向 Elasticsearch 中索引数据之前轻松地处理数据。

同时，Kibana 不仅可针对 Elasticsearch 数据提供实时可视化，同时还提供 UI 以便用户快速访问应用程序性能监测 (APM)、日志和基础设施指标等数据。

Elasticsearch 支持多种编程语言

- Java
- JavaScript (Node.js)
- Go
- .NET (C#)
- PHP
- Perl
- Python
- Ruby

适用场景

Elasticsearch 在速度和可扩展性方面都表现出色，而且还能够索引多种类型的内容，这意味着其可用于多种使用场景：

- 应用程序搜索
- 网站搜索
- 企业搜索
- 日志处理和分析
- 基础设施指标和容器监测

- 应用程序性能监测
- 地理空间数据分析和可视化
- 安全分析
- 业务分析

基本概念

本节列出有关 Elasticsearch 涉及的专有名词及术语。

- 节点 (node)

数据节点: 存储索引数据的节点，主要对文档进行增删改查、聚合等操作。

专有主节点: 对集群进行操作，例如创建或删除索引，跟踪哪些节点是集群的一部分，并决定哪些分片分配给相关的节点。稳定的主节点对集群的健康非常重要，默认情况下集群中的任一节点都可能被选为主节点。

协调节点: 分担数据节点的 CPU 开销，从而提高处理性能和服务稳定性。

- 索引 (Index)

用于存储 Elasticsearch 的数据，是一个或多个分片分组在一起的逻辑空间。

- 分片 (Shard)

索引可以存储数据量超过 1 个节点硬件限制的数据。为满足这样的需求，Elasticsearch 提供了一个能力，将一个索引拆分为多个，称为 Shard。

当您创建一个索引时，您可以根据实际情况指定 Shard 的数量。每个 Shard 托管在集群中的任意一个节点中，且每个 Shard 本身是一个独立的、全功能的“索引”。

Shard 的数量只能在创建索引前指定，且在索引创建成功后无法修改。

- 副本 (replicas)

replicas 是索引的备份，Elasticsearch 可以设置多个副本。写操作会先在主分片上完成，然后分发到副本分片上。

因为索引的主分片和副本分片都可以对外提供查询服务，所以副本能够提升系统的高可用性和搜索时的并发性能。但如果副本太多，也会增加写操作时数据同步的负担。

离线升级中间件 - ElasticSearch 模块

本页说明从[下载中心](../../../../download/index.md)下载中间件 - ElasticSearch 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `_mcamel_` 字样是中间件模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 `chart-syncer` 同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 `load-image.yaml`

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

==== "已安装 chart repo"

若当前环境已安装 `chart repo`, `chart-syncer` 也支持将 `chart` 导出为 `tgz` 文件。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
 url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```
==== "未安装 chart repo"
```

若当前环境未安装 `chart repo`, `chart-syncer` 也支持将 `chart` 导出为 `tgz` 文件，并存放

在指定路径。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: LOCAL
 path: /local-repo # chart 本地路径
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```

```

1. 执行同步镜像命令。

```
```shell
charts-syncer sync --config load-image.yaml
```

```

Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
```shell
tar -xvf mcamel-elasticsearch_0.10.1_amd64.tar
cd mcamel-elasticsearch_0.10.1_amd64
tar -xvf mcamel-elasticsearch_0.10.1.bundle.tar
```

```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```

==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

==== "通过 helm repo 升级"

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep elasticsearch
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-elasticsearch http://{harbor url}/chartrepo/{project}
```

```

1. 更新 helm 仓库。

```
```shell
```

```

helm repo update mcamel/mcamel-elasticsearch # helm 版本过低会导致失败，若失败
，请尝试执行 helm update repo

- 选择您想安装的版本（建议安装最新版本）。

```shell

```
helm search repo mcamel/mcamel-elasticsearch --versions

```

```none

| NAME | CHART VERSION | APP VERSION | DESCRIPTION |
|-----------------------------|---------------|-------------|-----------------------------|
| mcamel/mcamel-elasticsearch | 0.10.1 | 0.10.1 | A Helm chart for Kubernetes |

...

- 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell

```
helm get values mcamel-elasticsearch -n mcamel-system -o yaml > mcamel-elasticsearch.yaml

```

- 执行 `helm upgrade`。

升级前建议您覆盖 mcamel-elasticsearch.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```shell

```
export imageRegistry={你的镜像仓库}
---
```

```shell

```
helm upgrade mcamel-elasticsearch mcamel/mcamel-elasticsearch \
-n mcamel-system \
-f ./mcamel-elasticsearch.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.10.1

```

==== "通过 chart 包升级"

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-elasticsearch -n mcamel-system -o yaml > mcamel-elasticsearch.yaml
```
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址

。

```
```shell
export imageRegistry={你的镜像仓库}
```
```

```

```
```shell
helm upgrade mcamel-elasticsearch . \
-n mcamel-system \
-f ./mcamel-elasticsearch.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```
```

```

Elasticsearch 索引服务 Release Notes

本页列出 Elasticsearch 索引服务的 Release Notes，便于您了解各版本的演进路径和特性变化

。

*[mcamel-elasticsearch]: mcamel 是 DaoCloud 所有中间件的开发代号，elasticsearch 是提供分布式搜索和分析服务的中间件。

2025-02-28

v0.24.0

- **新增** 支持创建 `8.17.1` 版本的 `Elasticsearch` 实例

2024-11-30

v0.23.0

- **优化** 默认不启用 geoip 数据库以避免健康状态为 yellow

2024-09-30

v0.21.0

- **新增** 创建实例时支持选择 HTTPS / HTTP 协议
- **修复** 部分操作无审计日志的问题
- **修复** 安装器创建的 Elasticsearch 实例纳管失败的问题

2024-08-31

v0.20.0

- **优化** 创建实例时不可选择异常的集群

2024-07-31

v0.19.0

- **新增** 支持备份、恢复 Elasticsearch 实例
- **修复** 移除恢复实例的节点亲和性，并在恢复的集群中添加来源信息
- **修复** 在密码中含有特殊字符导致请求失败

2024-05-31

v0.17.0

- **修复** 容器列表资源使用率以限制量为分母
- **修复** exporter 无法正常展示资源利用率

2024-04-30

v0.16.0

- **优化** 增加命名空间配额的提示
- **修复** 一些内部错误

2024-03-31

v0.15.0

- **优化** 当用户权限不足时无法读取 elasticsearch 的密码

2024-01-31

v0.14.0

- **优化** Elasticsearch 实例支持中文 Dashboard
- **优化** 在全局管理中增加 Elasticsearch 版本展示

2023-12-31

v0.13.0

- **修复** 创建实例时部分输入框填写特殊字符的校验未生效的问题

2023-11-30

v0.12.0

- **新增** 支持记录操作审计日志
- **优化** 实例列表未获取到列表信息时的提示

2023-10-31

v0.11.0

- **新增** 离线升级
- **新增** 实例重启功能
- **修复** cloudshell 权限问题

2023-08-31

v0.10.0

- **优化** KindBase 语法兼容
- **优化** 在创建页面添加默认反亲和配置
- **优化** operator 创建过程的页面展示

2023-07-31

v0.9.3

- **新增** UI 界面的权限访问限制

2023-06-30

v0.9.0

- **新增** `_mcamel-elasticsearch_` 节点反亲和配置
- **新增** `_mcamel-elasticsearch_` 监控图表，去除干扰元素并新增时间范围选择
- **优化** `_mcamel-elasticsearch_` ServiceMonitor 闭环安装
- **修复** `_mcamel-elasticsearch_` 监控图表，去除干扰元素并新增时间范围选择

2023-05-30

v0.8.0

- **新增** `_mcamel-elasticsearch_` 新增对接全局管理审计日志模块
- **新增** `_mcamel-elasticsearch_` 新增可配置实例监控数据采集间隔时间
- **新增** `_mcamel-elasticsearch_` 修复 Pod 列表分页展示有误

2023-04-27

v0.7.2

- **新增** `_mcamel-elasticsearch_` 详情页面展示相关的事件
- **新增** `_mcamel-elasticsearch_` 支持自定义角色
- **优化** `_mcamel-elasticsearch_` 调度策略增加滑动按钮
- **修复** `_mcamel-elasticsearch_` 在纳管集群时可能会中断重试的问题

2023-03-28

v0.6.0

- **新增** `_mcamel-elasticsearch_` 支持中间件链路追踪适配
- **新增** 在安装 `_mcamel-elasticsearch_` 根据参数配置启用链路追踪
- **新增** `_mcamel-elasticsearch_` Kibana 支持 LoadBalancer 类型
- **升级** `golang.org/x/net` 到 v0.7.0
- **升级** GHippo SDK 到 v0.14.0

2023-02-23

v0.5.0

新功能

- **新增** `mcamel-elasticsearch` helm-docs 模板文件
- **新增** `mcamel-elasticsearch` 应用商店中的 Operator 只能安装在 mcamel-system
- **新增** `mcamel-elasticsearch` 支持 Cloud Shell
- **新增** `mcamel-elasticsearch` 支持导航栏单独注册
- **新增** `mcamel-elasticsearch` 支持查看日志
- **新增** `mcamel-elasticsearch` Operator 对接 chart-syncer
- **新增** `mcamel-elasticsearch` 支持 LB
- **新增** 日志查看操作说明，支持自定义查询、导出等功能

升级

- **升级** `mcamel-elasticsearch` 升级离线镜像检测脚本

修复

- **修复** `mcamel-elasticsearch` 实例名太长导致自定义资源无法创建的问题
- **修复** `mcamel-elasticsearch` 工作空间 Editor 用户无法查看实例密码
- **修复** `mcamel-elasticsearch` 密码不能使用特殊字符的问题
- **修复** `mcamel-elasticsearch` 超出索引导致 panic 的问题

2022-12-25

v0.4.0

新功能

- **新增** `mcamel-elasticsearch` 获取集群已经分配的 NodePort 列表接口
- **新增** `mcamel-elasticsearch` 增加状态详情
- **新增** `mcamel-elasticsearch` 节点亲和性配置

优化

- **优化** `mcamel-elasticsearch` 可以展示公共的 es，纳管之前是不可以删除的
- **优化** `mcamel-elasticsearch` 增加健康状态返回

修复

- **修复** `mcamel-elasticsearch` 修复 kb 不存在时候，删除会失败的 BUG
- **修复** `mcamel-elasticsearch` 修复 es exporter 离线失效的问题
- **修复** `mcamel-elasticsearch` 修复 es 创建成功后没有返回 ports 信息的 bug
- **修复** `mcamel-elasticsearch` 查询实例列表和详情时，Kibana 的服务类型不符合预期

2022-11-28

v0.3.6

- **改进** 密码校验调整为 MCamel 中等密码强度
- **改进** 角色可以升级
- **新增** 新增 sc 扩容提示
- **新增** 返回列表或者详情时的公共字段
- **新增** 返回告警
- **新增** 校验 Service 注释
- **修复** 更新实例后，集群使用了错误的镜像，导致集群状态异常
- **修复** 使用 NodePort 时，更新实例报错
- **升级** 中依赖的 eck operator 版本为 2.3.0
- **优化** 在某些版本的 K8s 集群中，默认 FD 不足，无法启动的问题
- **优化** 减小 elasticsearch 容器的运行权限

2022-10-28

v0.3.4

新功能

- **新增** 同步 pod 状态到实例详情页
- **新增** 获取用户列表接口
- **新增** 支持 arm 架构

优化

- **优化** workspace 界面逻辑调整
- **优化** 不符合设计规范的样式调整
- **优化** password 获取逻辑调整
- **优化** CPU 和内存请求量应该小于限制量逻辑调整
- **优化** 实例版本不允许修改，下拉框应该为文本

修复

- **修复** 更新实例服务设置，确认无反应，无法提交

2022-9-25

v0.3.2

- **新增** 列表页增加分页功能
- **新增** 增加修改配置的功能
- **新增** 增加返回可修改配置项的功能

- **新增** 更改创建实例的限制为集群级别，原来为 namespace 级别
- **新增** 增加监控地址的拼接功能
- **新增** 增加可以修改版本号的功能
- **新增** 修改底层 update 逻辑为 patch 逻辑
- **新增** 将时间戳 api 字段统一调整为 int64
- **新增** 单测覆盖率提升到 43%
- **新增** 对接全局管理增加 workspace 接口
- **新增** 对接 insight 通过 crd 注入 dashboard
- **新增** 更新 release note 脚本，执行 release-process 规范
- **新增** 支持 helm 部署 eck-operator
- **新增** 支持 helm 部署 mcamel-elasticsearch 服务
- **新增** 第一次文档网站发布
- **新增** 功能说明
- **新增** 产品优势
- **新增** 什么是 Elasticsearch
- **新增** 基本概念
- **新增** 集群容量规划

集群规格和容量规划

前言

当您准备创建 Elasticsearch 实例时，建议提前评估集群所需的资源。一般情况下需要评估所需磁盘容量、集群规格、存储的文件类型、大小和数量等。基于这些资源评估，预先规划合适的 Elasticsearch 集群配置，可以减少后续使用过程中的问题和风险。

> 重要知识: ES 集群的物理磁盘容量 **不等于** ES 集群可使用数据空间大小

本文基于实际测试结果和用户使用经验，提供了相对通用的评估方法。但业务是瞬息万变的，所以本文的目的是在您创建 Elasticsearch 时给出一个相对合适的建议。实际上 DCE 5 也提供了优化集群规格和容量规划的能力，方便后续对资源进行升级。

事先评估项

要进行 Elasticsearch 的容量规划，需要考虑以下因素：

- 使用场景：主要使用的业务场景，不同场景下产生的数据量不同
- 文档数量：确定要存储的文档数量，以及每个文档的大小
- 存储需求：确定所需的存储空间，以便存储所有文档以及任何相关数据
- 索引需求：选择适当的索引策略和设置以确保性能和可扩展性
- 查询需求：确定查询负载的复杂性和大小，以便为其分配足够的资源
- 集群规模：确定集群的大小和数量，以便支持预期的负载并提供高可用性和容错性

存储容量规划

基础要求规划

在存储容量规划的开始时，需要预先确认两个 使用场景 和 数据要求 规划：

- 使用场景

- 日志场景
- 搜索场景
- 数据分析场景
- 数据库加速场景
- 通用场景

- 数据要求

- 源数据大小或者预估文档条数和单个文档的大小
- 每日数据增量情况
- 数据保留时长
- 需要的副本数

ES 其他业务开销

Elasticsearch 中除了数据之外，还有其他的开销，这些也是影响 Elasticsearch 服务存储容量的主要原因，如：

- 索引开销：可以使用 cat/indices?v API 和 pri.store.size 值计算确切的开销计算，通常比源数据大 10%（all 参数等未计算）
- 操作系统预留空间：默认操作系统会保留 5% 的文件系统供您处理关键流程、系统恢复以及防止磁盘碎片化问题等
- Elasticsearch 内部开销：段合并、日志等内部操作，一般预留 20%
- 副本数量：副本有利于增加数据的可靠性，但同时会增加存储成本
- 安全阈值：为了预防突发的数据增长，请大致保留 15% 的容量空间作为安全阈值

所以，为了数据的安全和稳定，我们建议您的磁盘使用率不要超过 85%；或者在即将达到 85%，应该尽快扩充升级。

快速计算公式

完整公式 | 简化版本 |

| --- | --- |

| 源数据 * (1 + 副本数量) * (1 + 索引开销) / (1 - Linux 预留空间) / (1 - 内部开销) =
最小存储要求 | 源数据 * (1 + 副本数量) * 1.45 = 最小存储要求 |

如果有 500G 数据存储并且需要一个副本，则最低存储要求更接近 $500 * 2 * 1.1 / 0.95 = 1.5T$ 。

关于节点磁盘的配置

使用场景不同，单节点最大承载数据量也会不同，具体如下：

- 数据加速、查询聚合等场景：单节点磁盘最大容量 = 单节点内存大小 (GB) * 10
- 日志写入、离线分析等场景：单节点磁盘最大容量 = 单节点内存大小 (GB) * 50
- 通用场景：单节点磁盘最大容量 = 单节点内存大小 (GB) * 30

ES 集群实例配置推荐

在生产环境部署推荐配置：尽量一个节点只承担一个角色。不同节点所需要的计算资源不一样。不同角色分离后，可以按需扩展互不影响。

- 集群最大节点数 = 单节点 CPU * 5
- 单节点磁盘最大容量
 - 搜索类场景：单节点磁盘最大容量 = 单节点内存大小 (GB) * 10。
 - 日志类等场景：单节点磁盘最大容量 = 单节点内存大小 (GB) * 50。

配置 | 最大节点数 | 单节点磁盘最大容量 (查询) | 单节点磁盘最大容量 (日志)

- --|---|---|--

4 核 16G | 20 | 160 GB | 800 GB

8 核 32G | 40 | 320 GB | 1.5 TB

16 核 64G | 80 | 640 GB | 2 TB

分片数量规划

适用场景：

- 日志类，写入频繁，查询较少，单个分片 30G 左右
- 搜索类，写入少，查询频繁，单个分片不超过 20G

每个 Elasticsearch 索引被分为多个分片，数据按哈希算法打散到不同的分片中。由于索引分片的数量影响读写性能和故障恢复速度，建议提前规划。

分片使用概要

- Elasticsearch 在 7.x 版本中，每个索引默认为 1 个主分片 和 1 个副本分片
- 在单节点上，7.x 版本最大分片数量为 1000
- 单个分片大小尽量保持在 10-50G 之间为最佳体验，一般推荐在 30G 左右

- 分片过大可能使 `Elasticsearch` 的故障恢复速度变慢
 - 分片过小可能导致非常多的分片，因为每个分片会使用占用一些 CPU 和内存，从而导致读写性能和内存不足的问题。
-
- 当分片数量超过数据节点数量时，建议分片数量接近数据节点的整数倍，便于将分片均匀的分布到数据节点中。
 - 对日志场景，建议启用 ILM 功能。在发现分片大小不合理时，通过该功能及时调整分片数量。

索引分片资源占用

每个索引和每个分片都需要一些内存和 CPU 资源。在大多数情况下，一大组大分片比许多小分片使用更少的资源。

段在分片的资源使用中起着重要作用。大多数分片包含几个段，用于存储其索引数据。

`Elasticsearch` 将段元数据保存在 JVM 堆内存中，以便可以快速检索它以进行搜索。随着分片的增长，它的段被合并成更少、更大的段。这减少了段的数量，这意味着更少的元数据保存在堆内存中。

为了减少索引数量并避免造成过大且无序的映射，可以考虑在同一索引中存储类似结构的数据，而不要基于数据来源将数据分到不同的索引中。

很重要的一点是在索引/分片的数量和每个单独索引的映射大小之间实现良好平衡。

由于集群状态会加载到每个节点（包括主节点）上的堆内存中，而且堆内存大小与索引数量以及单个索引和分片中的字段数成正比关系，所以还需要同时监测主节点上的堆内存使用量并确保其大小适宜，这一点很重要。

分片过小会导致段过小，进而致使开销增加。您要尽量将分片的平均大小控制在至少几 GB 到几十 GB 之间。

对时序型数据用例而言，分片大小通常介于 20GB 至 40GB 之间。

由于单个分片的开销取决于段数量和段大小，所以通过 `forcemerge` 操作强制将较小的段合并为较大的段能够减少开销并改善查询性能。

理想状况下，应当在索引内再无数据写入时完成此操作。请注意：这是一个极其耗费资源的操作，所以应该在非高峰时段进行。

每个节点上可以存储的分片数量与可用的堆内存大小成正比关系，但是 Elasticsearch 并未强制规定固定限值。

这里有一个很好的经验法则：确保对于节点上已配置的每个 GB，将分片数量保持在 20 以下。

如果某个节点拥有 30GB 的堆内存，那其最多可有 600 个分片，但是在此限值范围内，您设置的分片数量越少，效果就越好。

一般而言，这可以帮助集群保持良好的运行状态。

更多信息，请参考：

- [减少集群分片数](<https://www.elastic.co/guide/en/elasticsearch/reference/7.17/size-your-shards.html#reduce-cluster-shard-count>)

- [我在 Elasticsearch 集群内应该设置多少个分片？](<https://www.elastic.co/cn/blog/how-many-shards-should-i-have-in-my-elasticsearch-cluster>)

分片计算公式

(元数据 + 增长空间) * (1 + 索引开销) / 所需的分片大小 = 主分片的大约数量

假设有 80GiB 的数据。希望将每个分片保持在 30GiB 左右。因此，您的分片数量应大约为 $80 * 1.1 / 30 = 3$

如何管理分片

进入到 Kibana 的索引管理界面，找到 Stack Management

使用索引生命周期管理（ILM）自动管理索引，管理策略如下：

- 根据索引大小，自动 rollover
- 根据索引创建时间，自动 rollover
- 根据文档数量，自动 rollover

索引生命周期执行策略，默认每 10 分钟执行一次，可以通过修改 `indices.lifecycle.poll_interval` 参数来控制检查频率。

数据服务内创建 Elasticsearch

> 中间件 elasticsearch 创建的页面

![创建实例](docs/zh/docs/middleware/elasticsearch/images/elasticsearch_storage.png)

节点介绍：

- **数据节点：** Data Node 是 Elasticsearch 用于存储数据的节点，负责存储数据，用于存储大量数据，所以对存储的压力非常高
 - **Kibana 节点：** 启用对应 Kibana 控制台，为 Elasticsearch 提供了界面化的管理窗口
 - **专用主节点：** 专用节点（Dedicated Node）是一种配置选项，该选项指定节点只用于执行特定任务。与数据节点（Data Node）不同，专用节点不存储任何数据，而是专门用于执行集群管理或搜索操作

如上图中的，存储容量配置，请根据上述的容量计算公式，规划对应的存储容量。是否启用专用主节点，主要考虑是否需要专用主节点来支持增强集群的稳定性。

结语

如果你有关于 Elasticsearch 如何进行容量规划的需求，可在页面底部进行讨论。

创建 Elasticsearch 实例

在 Elasticsearch 实例列表中，执行以下操作创建一个新的实例。

1. 在 Elasticsearch 搜索服务的首页右上角点击 新建实例。

![新建实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create01.png)

2. 填写实例的 基本信息。通过 安装环境检测 后，点击 下一步。

> 如未通过安装环境检测，页面会给出失败原因和操作建议。常见原因是缺少相关组件，根据页面提示安装对应的组件即可。

![基本信息](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/elasticsearch/images/create02.png)

3. 选择版本：选择基于哪个 Elasticsearch 版本创建实例，目前仅支持 7.16.3

![规格配置](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03.png)

4. 参考以下信息填写实例的 规格配置。

==== "数据节点（默认启用）"

- 用于存储数据，执行增删改查、搜索、聚合等数据相关操作。数据节点对资源要求较高，需要配置充足的资源。

- **如果未启用 专用主节点，由 数据节点 充当专用主节点**。
- 最少 1 副本，最多 50 个副本，默认 3 副本。
- 副本数建议为奇数，否则存在脑裂风险。

![热数据节点](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-1.png)

==== "Kibana 节点（默认启用）"

- Kibana 是适用于 Elasticsearch 的可视化分析平台，可搜索、查看存储在索引中的数据并与其进行交互。

- 默认启用 Kibana 节点，用于存放 Elasticsearch 可视化数据的节点。

- 默认为 1 副本，不可修改。

![Kibana 节点](<https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-2.png>)

==== "专用主节点"

- 主节点负责集群范围内的轻量化操作，例如创建/删除索引、监听其他类型节点的状态、决定如何分配数据分片等。

- **如果不启用 专用主节点，则由数据节点充当主节点**。这样可能会存在数据节点和主节点竞争资源的情况，影响系统稳定性。

- 启用 专用主节点 后，主节点与 数据节点 分离，有利于保障服务的稳定性。

- 默认 3 个副本，不可修改。

![专用主节点](<https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-3.png>)

==== "冷数据节点"

- 存储历史数据等查询频率低且基本无需写入的数据。

- 最少 2 副本，最多 50 副本，默认 3 副本。

- 如果业务中同时存在“查询频率高/写入压力大”和“查询频率/低基本无写入”的数据，建议启用 冷数据节点，实现冷热数据分离。

- 启用 冷数据节点 后，会自动开启 专用主节点。

![冷数据节点](<https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create03-4.png>)

5. 参考以下说明填写 服务设置

- 访问类型：Elasticsearch 实例对应的 Service 的类型。有关各种类型的详细说明，可参考[服务类型](<https://kubernetes.io/zh-cn/docs/concepts/services-networking/service/#publishing-services-service-types>)

- 访问设置：访问 Elasticsearch 实例的用户名和密码，以及 Kibana 的访问类型。

![服务设置](<https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/elasticsearch/images/create04.png>)

- 节点亲和性：启用后，只能/尽量将 Elasticsearch 实例调度到带有特定标签的节点上。
- 工作负载亲和性：在拓扑域（反亲和性的作用范围）内，根据反亲和性将工作负载下的 Pod 分发到多个节点中，避免多个 Pod 被集中调度到某一个节点，造成节点过载。相关的视频教程可参考[工作负载反亲和性]([../../../../videos/mcamel.md#_1](#))

- 监控采集时间间隔：实例监控的数据采集时间间隔。如果不设置，将采用全局设置。默

认为 30s。

![服务设置](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/elasticsearch/images/create04-1.png)

6. 检查所填信息，确认无误后点击 确定。如需修改可点击 上一步 返回修改配置。

![配置确认](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create05.png)

实例创建成功后，页面会自动跳转到 Elasticsearch 实例列表，可查看所有实例的基本信息和状态。

![创建成功](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/create06.png)

更新 Elasticsearch 实例

如果想要更新或修改 Elasticsearch 实例的资源配置，可以按照本页说明操作。

1. 在 Elasticsearch 实例列表中，点击右侧的 | 按钮，在弹出菜单中选择 更新实例。

![更新实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/update01.png)

2. 修改基本信息后，点击 下一步。此处暂时只支持修改描述信息。

![更新实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/update02.png)

3. 修改规格配置（包括热数据节点、Kibana 节点、专用主节点和冷数据节点）后，点击 下一步。

![更新实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/update03.png)

4. 修改服务设置后，点击 确定。

![更新实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/update04.png)

5. 返回消息队列，屏幕右上角将显示消息：更新实例成功。

![更新实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/update05.png)

删除 Elasticsearch 实例

如果想要删除一个 Elasticsearch 实例，可以执行如下操作：

!!! warning

删除实例后，该实例相关的所有信息也会被全部删除，请谨慎操作。

1. 在 Elasticsearch 实例列表中，点击右侧的 按钮，在弹出菜单中选择 删除实例。

![删除实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/delete01.png)

2. 在弹窗中输入该实例的名称，确认无误后，点击 删除 按钮。

![删除实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/delete03.png)

3. 自动返回实例列表，屏幕提示：删除实例 xxx 成功。

![删除实例](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/delete04.png)

实例概览

本文介绍 Elasticsearch 实例的基本信息。

1. 在 Elasticsearch 搜索服务 的首页点击实例名称

![概览](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/elasticsearch/images/list01.png)

2. 查看实例的概览信息，具体包括：

- 基本信息：实例名称、运行状态、部署位置、创建时间、版本、副本数、访问地址（Elasticsearch 集群内部的访问地址）等

- 健康状态：分为 Green、Red、Yellow

- 访问设置：访问方式、用户名和密码、Kibana 控制台地址等
- 资源配额：CPU 和内存使用率
- 监控告警：当前实例的告警信息，可以转到 [DCE 5.0 可观测性模块](../../../../insight/intro/index.md)查看告警详情或创建告警策略
- 容器组列表：当前实例的容器组信息，包括容器组名称、节点类型、运行状态、IP、CPU、内存、存储信息、创建时间等
- 最近事件：当前实例的事件信息

![概览](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/elasticsearch/images/basicinfo01.png)

实例监控

本页说明如何从图表面板快速获取 Elasticsearch 有关的信息。

1. 在 Elasticsearch 实例列表中点击一个实例名称。

![概览](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/elasticsearch/images/list01.png)

2. 在左侧导航栏点击 实例监控，进入实例监控大屏。

![概览](https://docs.daocloud.io/daocloud-docs-images/docs/zh/docs/middleware/elasticsearch/images/monitor01.png)

查看 Elasticsearch 日志

操作步骤

通过访问每个 Elasticsearch 的实例详情，页面；可以支持查看 Elasticsearch 的日志。

1. 在 Elasticsearch 实例列表中，选择想要查看的日志，点击 实例名称 进入到实例详情页面。

![image](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/ilog01.png)

2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。

![image](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/ilog02.png)

3. 点击 日志查看 即可进入到日志查看页面（[Insight](../../../../insight/intro/index.md) 日志查

看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- * 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- * 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- * 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- * 支持 查看日志的上下文，点击右侧 上下文 图标即可
- * 支持 导出日志

![image](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/elasticsearch/images/log03.png)

Elasticsearch 备份和恢复

Elasticsearch 的备份恢复功能可以帮助您备份整个集群或者单个索引的数据，以便在需要时进行恢复。您可以设置自动快照策略定期备份数据，或手动备份选择备份的索引。

实例的第一个备份快照是实例中数据的完整拷贝，后续所有的备份快照保留的是已存快照和新备份快照之间的增量，因此首次备份耗时较长（具体时长与数据量相关），后续快照备份会比较快。

备份配置和方法

配置备份路径

1. 点击左侧导航栏进入 备份管理 -> 备份设置。
2. 选择已添加工作空间中的 `S3 备份`，填写备份路径。

!!! info

手动备份和自动备份均依赖此处的备份配置。

![alt text](docs/zh/docs/middleware/elasticsearch/images/snapshot.png)

手动创建备份

1. 点击左侧导航栏进入 备份管理 -> 备份数据，查看该实例的已有备份数据。

![alt text](docs/zh/docs/middleware/elasticsearch/images/snapshot-1.png)

2. 点击列表右上角的创建备份按钮，填写备份名称、选择预备份的索引。

- 支持全选或多选备份的是业务索引。勾选系统索引后，默认备份全被的系统索引。

![alt text](docs/zh/docs/middleware/elasticsearch/images/snapshot-2.png)

开启自动备份

1. 点击左侧导航栏进入 备份管理 -> 备份设置。
2. 开启自动备份的开关，选择备份周期、备份时间以及自动备份保留的份数。

!!! info

自动备份默认仅备份所有的业务索引。

![alt text](docs/zh/docs/middleware/elasticsearch/images/snapshot-3.png)

恢复备份数据

1. 点击左侧导航栏进入 备份管理 -> 备份数据，选择想要恢复的备份数据，点击操作列的 |，点击恢复。

![alt text](docs/zh/docs/middleware/elasticsearch/images/snapshot-4.png)

2. 填写恢复后的实例名称，并选择恢复到当前实例的位置或选择恢复到其他集群、命名空间。

![alt text](docs/zh/docs/middleware/elasticsearch/images/snapshot-5.png)

Elasticsearch 排障手册

本文将持续统计和梳理常见的 Elasticsearch 异常故障以及修复方式。若遇到使用问题，请优先查看此排障手册。

> 如果您发现遇到的问题未包含在本手册，可以快速跳转到页面底部，提交您的问题。

Elasticsearch PVC 磁盘容量满

> 存储依赖 hwameistor

报错信息

```
```info
{
 "type": "server", "timestamp": "2022-12-18T10:47:08,573Z", "level": "ERROR", "component": "o.e.m.FsHealthService", "cluster.name": "mcamel-common-es-cluster-masters", "node.name": "mc amel-common-es-cluster-masters-es-masters-0", "message": "health check of [/usr/share/elasticsearc h/data/nodes/0] failed", "cluster.uuid": "afIlgTVTXmYO2qPFNvsuA", "node.id": "nZRiBCUZQy mQVV1son34pA" ,
 "stacktrace": ["java.io.IOException: No space left on device",
 "at sun.nio.ch.FileDispatcherImpl.write0(Native Method) ~[?: ?]",
 "at sun.nio.ch.FileDispatcherImpl.write(FileDispatcherImpl.java: 62) ~[?:?]",
 "at sun.nio.ch.IOUtil.writeFromNativeBuffer(IOUtil.java: 132) ~[?:?]",
 "at sun.nio.ch.IOUtil.write(IOUtil.java: 97) ~[?:?]",
 "at sun.nio.ch.IOUtil.write(IOUtil.java: 67) ~[?:?]",
 "at sun.nio.ch.FileChannelImpl.write(FileChannelImpl.java: 285) ~[?:?]",
 "at java.nio.channels.Channels.writeFullyImpl(Channels.java: 74) ~[?:?]",
 "at java.nio.channels.Channels.writeFully(Channels.java: 96) ~[?:?]",
 "at java.nio.channels.Channels$1.write(Channels.java: 171) ~[?:?]",
 "at java.io.OutputStream.write(OutputStream.java: 127) ~[?:?]",
 "at org.elasticsearch.monitor.fs.FsHealthService$FsHealthMonitor.monitorFSHealth(FsHealthService.java: 170) [elasticsearch-7.16.3.jar:7.16.3]",
 "at org.elasticsearch.monitor.fs.FsHealthService$FsHealthMonitor.run(FsHealthService.java: 144) [elasticsearch-7.16.3.jar:7.16.3]",
 "at org.elasticsearch.threadpool.Scheduler$ReschedulingRunnable.doRun(Scheduler.java: 214) [elasticsearch-7.16.3.jar:7.16.3]",
 "at org.elasticsearch.common.util.concurrent.ThreadContext$ContextPreservingAbstractRunnable.doRun(ThreadContext.java: 777) [elasticsearch-7.16.3.jar:7.16.3]",
 "at org.elasticsearch.common.util.concurrent.AbstractRunnable.run(AbstractRunnable.java: 26) [elasticsearch-7.16.3.jar:7.16.3]",
 "at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java: 1136) [?:?]",
 "at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java: 635) [?:?]",
 "at java.lang.Thread.run(Thread.java: 833) [?:?]"]
}
```

## 解决方式

### 1. 扩容 PVC (从 1Gi 修改为 10Gi)

```
kubectl edit pvc elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0 -n mcam el-system
spec:
 accessModes:
 - ReadWriteOnce
 resources:
 requests:
 storage: 10Gi
```

## 2. PVC 扩容日志

查看 `elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0` 扩容日志信息。

```
kubectl describe pvc.elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0 -n mcamel-system
Name: elasticsearch-data-mcamel-common-es-cluster-masters-es-masters-0
Namespace: mcamel-system
StorageClass: hwameistor-storage-lvm-hdd
Status: Bound
Volume: pvc-42309e19-b74f-45b4-9284-9c68b7dd93b3
Labels: common.k8s.elastic.co/type=elasticsearch
 .elasticsearch.k8s.elastic.co/cluster-name=mcamel-common-es-cluster-masters
 .elasticsearch.k8s.elastic.co/statefulset-name=mcamel-common-es-cluster-masters-es-masters
Annotations: pv.kubernetes.io/bind-completed: yes
 pv.kubernetes.io/bound-by-controller: yes
 volume.beta.kubernetes.io/storage-provisioner: lvm.hwameistor.io
 volume.kubernetes.io/selected-node: xulongju-worker03
Finalizers: [kubernetes.io/pvc-protection]
Capacity: 10Gi
Access Modes: RWO
VolumeMode: Filesystem
Used By: mcamel-common-es-cluster-masters-es-masters-0
Events:
 Type Reason Age From
 ---- ---- -- --
 Normal WaitForPodScheduled 51m (x18 over 55m) persistentvolume-controller
 waiting for pod mcamel-common-es-cluster-masters-es-masters-0 to be scheduled
 Normal WaitForFirstConsumer 50m (x7 over 56m) persistentvolume-controller
 waiting for first consumer to be created before binding
 Normal ExternalProvisioning 50m persistentvolume-controller
 waiting for a volume to be created, either by external provisioner
```

```

 "lvm.hwameistor.io" or manually created by system administrator
 Normal Provisioning 50m lvm.hwameistor.i
o_hwameistor-local-storage-csi-controller-68c9df8db8-kzdgn_680380b5-fc4d-4b82-ba80-568
1e99a8711 External provisioner is provisioning volume for claim "mcamel-system/elast
icsearch-data-mcamel-common-es-cluster-masters-es-masters-0"
 Normal ProvisioningSucceeded 50m lvm.hwameistor.io
 _hwameistor-local-storage-csi-controller-68c9df8db8-kzdgn_680380b5-fc4d-4b82-ba80-5681
e99a8711 Successfully provisioned volume pvc-42309e19-b74f-45b4-9284-9c68b7dd93b
3
 Warning ExternalExpanding 3m39s volume_expand

 Ignoring the PVC: didn't find a plugin capable of expanding
the volume; waiting for an external controller to process this PVC.
 Warning VolumeResizeFailed 3m39s external-resizer 1
vm.hwameistor.io
 resize volume "pvc-42309e19-b74f-45b4-9284-9c68b7dd93b3" by re
sizer "lvm.hwameistor.io" failed: rpc error: code = Unknown desc = volume expansion
not completed yet
 Warning VolumeResizeFailed 3m39s external-resizer 1
vm.hwameistor.io
 resize volume "pvc-42309e19-b74f-45b4-9284-9c68b7dd93b3" by re
sizer "lvm.hwameistor.io" failed: rpc error: code = Unknown desc = volume expansion
in progress
 Normal Resizing 3m38s (x3 over 3m39s) external-resizer 1v
m.hwameistor.io
 External resizer is resizing volume pvc-42309e19-b74f-45b4-9284-9c
68b7dd93b3
 Normal FileSystemResizeRequired 3m38s external-resizer 1v
m.hwameistor.io
 Require file system resize of volume on node
 Normal FileSystemResizeSuccessful 2m42s kubelet

```

## Elasticsearch 业务索引别名被占用

现象：索引别名被占用

image

image

此图中 `__*-write__` 为别名，例如 `jaeger-span-write`，需要对此别名进行处理

查看业务索引模板中使用的别名 `rollover_alias` 对应值

image

image

临时处理方式：进入 es pod 容器内执行以下脚本：

1. 修改 TEMPLATE\_NAME 对应值
2. 修改 INDEX\_ALIAS 对应值
3. 需要进入 elasticsearch pod 中执行该脚本
4. 修改里面 elastic 用户的密码值 (ES\_PASSWORD=xxxx)

```
#!/bin/bash
Add a template/policy/index
TEMPLATE_NAME=insight-es-k8s-logs
INDEX_ALIAS="${TEMPLATE_NAME}-alias"
ES_PASSWORD="DaoCloud"
ES_URL=https://localhost:9200
while [["$(curl -s -o /dev/null -w "%{http_code}\n" -u elastic: ${ES_PASSWORD} ${ES_URL} -k)" != "200"]]; do sleep 1; done
curl -XDELETE -u elastic: ${ES_PASSWORD} -k "${ES_URL}/${INDEX_ALIAS}"
curl -PUT -u elastic:${ES_PASSWORD} -k "${ES_URL}/${TEMPLATE_NAME}-000001" -H \
Content-Type: application/json' -d'{"aliases": {"\"${INDEX_ALIAS}\": {"is_write_index": true }}'
}'
```

注意：此脚本存在一定失败几率，取决于数据写入速度，作为临时解决方式。

真实情况需要停止数据源的写入情况，再执行上述方法。

## 报错 Error setting GoMAXPROCS for operator

### 报错信息

image

image

环境信息：

kind 版本: 0.17.0

containerd: 1.5.2

k8s: 1.21.1

### 解决方式

升级版本：

kind: 1.23.6

runc version 1.1.0

# 报错 Terminating due to java.lang.OutOfMemoryError: Java heap space

完整的报错信息如下：

```
{"type": "server", "timestamp": "2023-01-04T14:44:05,920Z", "level": "WARN", "component": "o.e.d.PeerFinder", "cluster.name": "gsc-cluster-1-master-es", "node.name": "gsc-cluster-1-master-es-es-data-0", "message": "address [127.0.0.1:9305], node [null], requesting [false] connection failed: [[127.0.0.1:9305] connect_exception: Connection refused: /127.0.0.1:9305: Connection refused", "cluster.uuid": "JOa0U_Q6T7WT60SPYiR1Ig", "node.id": "_zlorWVeRbyrUMYf9wJgfQ" } } {"type": "server", "timestamp": "2023-01-04T14:44:06,379Z", "level": "WARN", "component": "o.e.m.j.JvmGcMonitorService", "cluster.name": "gsc-cluster-1-master-es", "node.name": "gsc-cluster-1-master-es-es-data-0", "message": "[gc][15375] overhead, spent [1.3s] collecting in the last [1.3s]", "cluster.uuid": "JOa0U_Q6T7WT60SPYiR1Ig", "node.id": "_zlorWVeRbyrUMYf9wJgfQ" } {"timestamp": "2023-01-04T14:44:06+00:00", "message": "readiness probe failed", "curl_rc": "28"} } java.lang.OutOfMemoryError: Java heap space Dumping heap to data/java_pid7.hprof ... {"timestamp": "2023-01-04T14:44:11+00:00", "message": "readiness probe failed", "curl_rc": "28"} } {"timestamp": "2023-01-04T14:44:14+00:00", "message": "readiness probe failed", "curl_rc": "28"} } {"timestamp": "2023-01-04T14:44:17+00:00", "message": "readiness probe failed", "curl_rc": "28"} } {"timestamp": "2023-01-04T14:44:21+00:00", "message": "readiness probe failed", "curl_rc": "28"} } {"timestamp": "2023-01-04T14:44:26+00:00", "message": "readiness probe failed", "curl_rc": "28"} } {"timestamp": "2023-01-04T14:44:31+00:00", "message": "readiness probe failed", "curl_rc": "28"} } Heap dump file created [737115702 bytes in 25.240 secs] Terminating due to java.lang.OutOfMemoryError: Java heap space
```

## 解决方式

如果在条件允许的情况下，可以进行资源及容量规划。

```
kubectl edit elasticsearch mcamel-common-es-cluster-masters -n mcamel-system
image
image
```

## OCP 环境安装 Elasticsearch 时报错 Operation not permitted

### 报错信息

image

image

### 解决方式

image

image

## 某个节点磁盘读吞吐异常、CPU workload 很高

### 异常信息

image

image

image

image

### 解决方式

如果 es 在此节点，可以将 ES 进程杀掉恢复。

## 数据写入 Elasticsearch 时报错 status: 429

### 完整的报错信息如下：

```
[2023/03/23 09:47:16] [error] [output:es:es.kube.kubeevent.syslog] error: Output
{"took": 0,"errors":true,"items":[{"create":{"_index":"insight-es-k8s-logs-000067","_type":"_doc","_id":"MhomDIcBLVS7yRloG6PF","status":429,"error":{"type":"es_rejected_execution_exception","reason":"rejected execution of org.elasticsearch.action.support.replication.TransportWriteAction$1/WrappedActionListener{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7002/0x0000000801b2b3d0@16e9faf7}{org.elasticsearch.action.support.replication.ReplicationOperation$$Lambda$7003/0x0000000801b2b5f8@46bcb787} on EsThreadPoolExecutor[name = mcamel-common-es-cluster-masters-es-data-0/write, queue capacity = 10000, org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@499b0f50[Running, pool size = 2, active threads = 2, queued tasks = 10000, completed tasks = 11472149]]}}},{"create":{"_index":"insight-es-k8s-logs-000067","_type":"_doc","_id":"MxomDIcBLVS7yRloG6PF","status":429,"error":{"type":"es_rejected_execution_exception","reason":"rejected execution of org.elasticsearch.action.support.replication.TransportW
```

```

riteAction$1/WrappedActionListener{org.elasticsearch.action.support.replication.ReplicationOperation
$$Lambda$7002/0x0000000801b2b3d0@16e9faf7}{org.elasticsearch.action.support.replication.Replic
ationOperation$$Lambda$7003/0x0000000801b2b5f8@46bcb787} on EsThreadPoolExecutor[name =
mcamel-common-es-cluster-masters-es-data-0/write, queue capacity = 10000, org.elasticsearch.c
ommon.util.concurrent.EsThreadPoolExecutor@499b0f50[Running, pool size = 2, active threads =
2, queued tasks = 10000, completed tasks = 11472149]]}}}, {"create": {"_index": "insight-es-k8s
-logs-000067", "_type": "_doc", "_id": "NBomDIcBLVS7yRloG6PF", "status": 429, "error": {"type": "es_rej
ected_execution_exception", "reason": "rejected execution of org.elasticsearch.action.support.replicati
on.TransportWriteAction$1/WrappedActionListener{org.elasticsearch.action.support.replication.Repli
cationOperation$$Lambda$7002/0x0000000801b2b3d0@16e9faf7}{org.elasticsearch.action.support.r
eplication.ReplicationOperation$$Lambda$7003/0x0000000801b2b5f8@46bcb787} on EsThreadPoo
lExecutor[name = mcamel-common-es-cluster-masters-es-data-0/write, queue capacity = 10000, or
g.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@499b0f50[Running, pool size = 2,
active threads = 2, queued tasks = 10000, completed tasks = 11472149]]}}}}

```

## 解决方式

- 方式 1：产生 429 错误的原因是 **Elasticsearch** 写入并发过大，**Elasticsearch** 来不及处理导致，可以适当降低写入并发并控制写入量。
- 方式 2：在资源允许的情况下，可以适当调大队列大小

nodeSets:

```

- config:
 node.store.allow_mmap: false
 thread_pool.write.queue_size: 1000 #增加/调大此参数的值

```

方式 1 和方式 2 可以配合使用。

# 双机房部署 Elasticsearch

## 背景

假设 es 按照下面的拓扑结构部署，可能存在一个问题：索引分片的（2个）副本可能全部落在 zone-1，当 zone-1 发生灾难，则该分片的数据全部丢失。需要利用 es 自身的配置让分片的副本分布在 zone-1 与 zone-2。

## 操作步骤

1. 登录控制台，给集群中位于不同机房的 节点 打 label。假设 work1 work2 在一个机

房，work3 在另一个机房：

```
[root@prod-master2 ~]# kubectl get node -Lzone
NAME STATUS ROLES AGE VERSION ZONE
prod-master1 Ready control-plane 7h27m v1.27.5
prod-master2 Ready control-plane 7h27m v1.27.5
prod-master3 Ready control-plane 7h26m v1.27.5
prod-worker1 Ready <none> 7h24m v1.27.5 zone-1
prod-worker2 Ready <none> 7h24m v1.27.5 zone-1
prod-worker3 Ready <none> 7h24m v1.27.5 zone-2
00
```

2. es 的配置主要分为 2 部分，一部分是需要对 es 节点调度进行相应配置；二是需要

对 es 的配置文件进行相关配置

## 节点调度配置

- 在 es 的 yaml 中，我们需要配置 2 个 nodeSets（一个 nodeSet 对应一个 statefulset）

```
[root@prod-master2 ~]# kubectl -n mcamel-system get sts -l common.k8s.elastic.co/type=elasticsearch
NAME READY AGE
mcamel-common-es-cluster-masters-es-data-zone1 2/2 4h17m
mcamel-common-es-cluster-masters-es-data-zone2 1/1 4h17m
```

00

- 利用亲和性与反亲和配置，将其中两个 es 节点分布在 zone1，另一个 es 节点分布在 zone2：

```
[root@prod-master2 ~]# kubectl -n mcamel-system get pod -l common.k8s.elastic.co/type=elasticsearch -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
mcamel-common-es-cluster-masters-es-data-zone1-0 1/1 Running 0 4h19m 10.233.95.69 prod-worker2 <none> <none>
mcamel-common-es-cluster-masters-es-data-zone1-1 1/1 Running 0 4h19m 10.233.93.197 prod-worker1 <none> <none>
mcamel-common-es-cluster-masters-es-data-zone2-0 1/1 Running 0 4h18m 10.233.126.69 prod-worker3 <none> <none>
```

00

## 配置文件配置

将 zone-1 与 zone-2 分为两个 nodeSet 后，就可以对他们的配置文件分开配置，相关的配置有：

- cluster.routing.allocation.awareness.attributes
- node.attr.zone
- cluster.routing.allocation.awareness.force.zone.values

1. 对于 zone-1 的节点配置为：

```
node.attr.zone: zone-1
cluster.routing.allocation.awareness.attributes: zone
cluster.routing.allocation.awareness.force.zone.values: zone-1,zone-2
```

2. 对于 zone-2 的节点配置为：

```
node.attr.zone: zone-2
cluster.routing.allocation.awareness.attributes: zone
cluster.routing.allocation.awareness.force.zone.values: zone-1,zone-2
```

# 基于 Hwameistor 的 Elasticsearch 迁移实践

由于 Kubernetes 自身的特性，有状态应用部署完成后是否可以迁移取决于底层 CSI 的能力。然而当集群出现资源不均等意外情况时，需要跨节点迁移相关的有状态应用。

本文以 Elasticsearch 为例，参考 [Hwameistor](#) 官方提供的迁移指南，演示使用 Hwameistor 时如何跨节点迁移数据服务中间件。

## 演示环境

从集群信息、ES 安装信息、PVC 三方面进行介绍演示环境：

==== “集群信息”

```
```bash
[root@prod-master1 ~]# kubectl get node
NAME        STATUS   ROLES          AGE    VERSION
prod-master1 Ready    control-plane 15h    v1.25.4
prod-master2 Ready    control-plane 15h    v1.25.4
prod-master3 Ready    control-plane 15h    v1.25.4
prod-worker1 Ready    <none>         15h    v1.25.4
prod-worker2 Ready    <none>         15h    v1.25.4
```

```
prod-worker3    Ready    <none>        15h    v1.25.4
```

```

==== “ES 安装信息”

```
```bash
[root@prod-master1 ~]# kubectl get pods -o wide | grep es-cluster-masters-es-data
```

```
mcamel-common-es-cluster-masters-es-data-0 Running prod-worker1
mcamel-common-es-cluster-masters-es-data-1 Running prod-worker3
mcamel-common-es-cluster-masters-es-data-2 Running prod-worker2
```

```

==== “ES 使用的 PVC”

```
```bash
kubectl -n mcamel-system get pvc -l elasticsearch.k8s.elastic.co/statefulset-name=mcamel-common
-es-cluster-masters-es-data
NAME                           STATUS   VOLU
ME                            CAPACITY   ACCESS MODES   STORAGEC
LASS                           AGE
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-0   Bound   pvc-61776435-0df5-44
8f-abb9-4d06774ec0e8   35Gi      RWO          hwameistor-storage-lvm-hdd   15h
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1   Bound   pvc-7d4c45c9-49d6-46
84-aca2-8b853d0c335c   35Gi      RWO          hwameistor-storage-lvm-hdd   15h
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-2   Bound   pvc-955bd221-3e83-4
bb5-b842-c11584bcfd10   35Gi      RWO          hwameistor-storage-lvm-hdd   15h
```

```

## 演示目标

将 **prod-worker3** 节点上的 **mcamel-common-es-cluster-masters-es-data-1** (以下简称 **演示应用 / esdata-1** ) 有状态应用跨节点迁移到 **prod-master3** 节点。

## 准备工作

### 确定需要迁移的 PV

使用如下命令查找演示应用 **esdata-1** 对应的 **PV** 磁盘，明确需要迁移哪个 **PV**。

1. 查看演示应用绑定的 PVC

```
[root@prod-master1 ~]# kubectl -n mcamel-system get pod mcamel-common-es-cluster-masters-es-data-1 -ojson | jq .spec.volumes[0]
{
 "name": "elasticsearch-data",
 "persistentVolumeClaim": {
 "claimName": "elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1"
 }
}
```

## 2. 查看该 PVC 绑定的 PV

```
[root@prod-master1 ~]# kubectl -n mcamel-system get pvc elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1
NAME STATUS
VOLUME CAPACITY ACCESS MODES
STORAGECLASS AGE
elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1 Bound pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 35Gi RWO hwameistor-storage-lvm-hdd 17h
```

## 3. 确认该 PV 绑定的应用是否为需要迁移的应用，即此文中的演示应用 esdata-1

```
[root@prod-master1 ~]# kubectl -n mcamel-system get pv
pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c NAME CAPACITY ACCESS MODES RECLAIM
POLICY STATUS CLAIM STORAGECLASS REASON AGE
pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 35Gi RWO Delete Bound
mcamel-system/elasticsearch-data-mcamel-common-es-cluster-masters-es-data-1
hwameistor-storage-lvm-hdd 17h ````
```

上述信息证明，需要迁移的 PV 为 pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c。

## 停止运行待迁移应用

### 1. 查看当前正在运行的应用

```
[root@prod-master1 ~]# kubectl -n mcamel-system get sts
NAME READY AGE
elastic-operator 2/2 20h
mcamel-common-es-cluster-masters-es-data 3/3 20h
mcamel-common-kpanda-mysql-cluster-mysql 2/2 20h
mcamel-common-minio-cluster-pool-0 1/1 20h
mcamel-common-mysql-cluster-mysql 2/2 20h
mysql-operator 1/1 20h
rfr-mcamel-common-redis-cluster 3/3 20h
```

### 2. 停止运行 ES operator

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=0 sts elastic-operator
```

### 3. 停止运行 ES :

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=0 sts mcamel-common-es-cluster-masters-es-data
--- wait about 3 mins ---
```

### 4. 确认 ES 已经停止运行

| NAME                                     | READY | AGE |
|------------------------------------------|-------|-----|
| elastic-operator                         | 0/0   | 20h |
| mcamel-common-es-cluster-masters-es-data | 0/0   | 20h |
| mcamel-common-kpanda-mysql-cluster-mysql | 2/2   | 20h |
| mcamel-common-minio-cluster-pool-0       | 1/1   | 20h |
| mcamel-common-mysql-cluster-mysql        | 2/2   | 20h |
| mysql-operator                           | 1/1   | 20h |
| rfr-mcamel-common-redis-cluster          | 3/3   | 20h |

视频演示如下：

[asciicast](#)

## 开始迁移

有关此过程的详细说明，可参考 **HWAMEISTOR** 官方文档：[迁移数据卷](#)

### 1. 建立迁移任务

```
[root@prod-master1 ~]# cat migrate.yaml
apiVersion: hwameistor.io/v1alpha1
kind: LocalVolumeMigrate
metadata:
 namespace: hwameistor
 name: migrate-es-pvc # 任务名称
spec:
 sourceNode: prod-worker3 # 来源 node，可以通过 `kubectl get ldn` 获取
 targetNodesSuggested:
 - prod-master3
 volumeName: pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c # 需要迁移的 pvc
 migrateAllVols: false
```

### 2. 执行迁移命令

```
[root@prod-master1 ~]# kubectl apply -f migrate.yaml
```

此时会在 **hwameistor** 命名空间创建一个 **pod**，用于执行迁移动作。

### 3. 查看迁移状态

```
[root@prod-master1 ~]# kubectl get localvolumemigrates.hwameistor.io migrate-es-pvc -o yaml
aml
apiVersion: hwameistor.io/v1alpha1
kind: LocalVolumeMigrate
metadata:
 annotations:
 kubectl.kubernetes.io/last-applied-configuration: |
 {"apiVersion":"hwameistor.io/v1alpha1","kind":"LocalVolumeMigrate","metadata":{},"annotations":{},"name":"migrate-es-pvc"},"spec":{"migrateAllVols":false,"sourceNode":"prod-worker3","targetNodesSuggested":["prod-master3"],"volumeName":"pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c"}}
 creationTimestamp: "2023-04-30T12:24:17Z"
 generation: 1
 name: migrate-es-pvc
 resourceVersion: "1141529"
 uid: db3c0df0-57b5-42ef-9ec7-d8e6de487767
 spec:
 abort: false
 migrateAllVols: false
 sourceNode: prod-worker3
 targetNodesSuggested:
 - prod-master3
 volumeName: pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c
 status:
 message: 'waiting for the sync job to complete: migrate-es-pvc-datacopy-elasticsearch-data-mcamel'
 originalReplicaNumber: 1
 state: SyncReplica
 targetNode: prod-master3
```

### 4. 迁移完成后，查看迁移结果

```
[root@prod-master1 ~]# kubectl get lvr
 NAME CAPACITY NODE STATE SYNCED DEVICE AGE
 pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 37580963840 prod-master3 Ready true /dev
 /LocalStorage_PoolHDD/pvc-7d4c45c9-49d6-4684-aca2-8b853d0c335c 129s
```

## 恢复 common-es

### 1. 启动 ES operator

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=2 sts elastic-operator
```

## 2. 启动 ES

```
[root@prod-master1 ~]# kubectl -n mcamel-system scale --replicas=3 sts mcamel-common-es-cluster-masters-es-data
```

## 相关问题

HwameiStor 使用 rclone 来迁移 PV，而 rclone 在迁移过程中可能会丢失权限（参考 [rclone#1202](#) 和 [hwameistor#830](#)）。如果权限丢失，ES 会启动失败并反复启动，陷入恶性循环。

遇到类似问题时可以通过下述步骤排查并解决故障。

## 确认问题

使用以下命令查看 Pod 日志：

```
```bash kubectl -n mcamel-system logs mcamel-common-es-cluster-masters-es-data-0 -c elasticsearch
```

如果日志中包含如下错误信息，则可以确认为权限丢失造成的问题。

```
```log
java.lang.IllegalStateException: failed to obtain node locks, tried [[/usr/share/elasticsearch/data]]
with lock id [0]; maybe these locations are not writable or multiple nodes were started without
increasing [node.max_local_storage_nodes] (was [1])?
```

## 解决故障

### 1. 运行下命令修改 ES 的 CR

```
kubectl -n mcamel-system edit elasticsearches.elasticsearch.k8s.elastic.co mcamel-common-es-cluster-masters
```

### 2. 为 ES 的 Pod 添加一个 `initcontainer`，内容如下：

- `command:`
  - `sh`
  - `-c`
  - `chown -R elasticsearch:elasticsearch /usr/share/elasticsearch/data`

```
 name: change-permission
 resources: {}
 securityContext:
 privileged: true
```

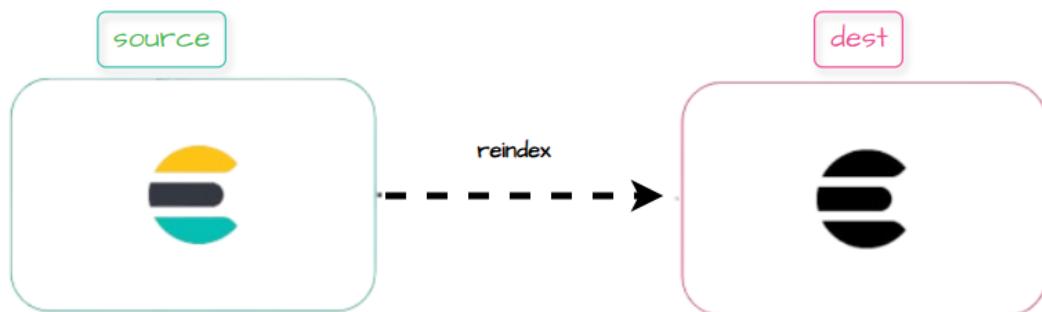
**initcontainer** 在 CR 中的位置如下：

```
spec:
 ...
 ...
 nodeSets:
 - config:
 node.store.allow_mmap: false
 count: 3
 name: data
 podTemplate:
 metadata: {}
 spec:
 ...
 ...
 initContainers:
 - command:
 - sh
 - -c
 - sysctl -w vm.max_map_count=262144
 name: sysctl
 resources: {}
 securityContext:
 privileged: true
 - command:
 - sh
 - -c
 - chown -R elasticsearch:elasticsearch /usr/share/elasticsearch/data
 name: change-permission
 resources: {}
 securityContext:
 privileged: true
```

# 使用 reindex 迁移 Elasticsearch 数据

## 背景

使用 Elasticsearch 的 reindex 功能实现跨集群之间的数据迁移，同时保证 source 集群业务不中断。为了完全同步 source 集群的数据，在某个时刻还是需要停止 source 集群的写入操作。



move-data-to-new-es

## 注意事项

- 重新索引要求源中的所有文档启用 `_source`。
- 在调用 `_reindex` 之前，应该将目标配置为所需的状态。重新索引不会复制源或其关联模板的设置。映射、分片数量、副本等必须提前配置好。
- 因为 `reindex` 有个 `snapshot` 的动作，在 `reindex` 过程中发生的数据变更是不会体现 在 `dest` 集群的。参见“source 集群的数据发生变化”这一节。

## 测试环境说明

### 虚拟机部署 Elasticsearch

- 版本：7.15.2

- 访问地址 : 172.30.120.85: 9200
- 用户名/密码 : elastic/root123!

## DCE 5.0 部署 Elasticsearch 实例

- 版本 : 7.16.3
- ES 访问地址 : https://10.6.178.179:30486
- 用户名/密码 : elastic/m5!586LM33Hz0qf

## 操作步骤

1. 修改 DCE 5.0 ES 的 CR 内容，将虚拟机 ES 的访问地址添加到白名单，如下图所示。

nodeSets:

```
- config:
 node.roles:
 - data
 - master
 - ingest
 - ml
 - data_cold
 - data_content
 - data_frozen
 - data_hot
 - data_warm
 - remote_cluster_client
 - transform
 reindex.remote.whitelist: 172.30.120.85:9200, localhost:* /添加虚拟机 ES 访问
地址
```

2. 查看虚拟机 ES 中的索引数据。

```
GET cat/indices/test_data*
```

```
yellow open test_data3 WA6qaRp5QC21nvuJMF33kA 1 1 10000 0 954.1kb 954.1kb
yellow open test_data4 _H8LWZ28RdmdVkJae-3Piw 1 1 10000 0 953.8kb 953.8kb
yellow open test_data1 xoFHZRvxT1uRMZ0tWxFfNQ 1 1 9000 0 869.3kb 869.3kb
yellow open test_data2 dtM5J7poS6Wo7BhymKaxcQ 1 1 10000 0 953.1kb 953.1kb
```

## 使用 reindex 迁移单个索引

- 在 DCE 5.0 ES 执行以下命令：

```
POST _reindex
{
 "source": {
 "remote": {
 "host": "http://172.30.120.85:9200",
 "username": "elastic",
 "password": "root123!"
 },
 "index": "test_data1"
 },
 "dest": {
 "index": "test_data1"
 }
}
```

- 迁移完成后，在 DCE 5.0 ES 中查询迁移的索引数据。

```
GET _cat/indices/test_data1
```

```
yellow open test_data1 46SIN0ddTDyKUeX_fzLwAw 1 1 9000 0 862.5kb 862.5kb
```

## 使用 reindex 迁移多个索引

```
#!/bin/bash
for index in 2 3 4; do
 curl -H "Content-Type: application/json" -k -u elastic:'m5!586LM33Hz0qf' -XPOST "https://10
.6.178.179:30486/_reindex?pretty" -d '{
 "source": {
 "remote": {
 "host": "http://172.30.120.85:9200",
 "username": "elastic",
 "password": "root123!"
 },
 "index": "test_data""$index"""
 },
 "dest": {
 "index": "test_data""$index"""
 }
}'
```

```

}'
done

```

## 使用异步 reindex 迁移数据

1. 请求的 url 加上 `wait_for_completion=false` 会立即返回，通过查看 `task` 来知道任务的进度。

```
Collapse source
POST _reindex?wait_for_completion=false
{
 "source": {
 "remote": {
 "host": "http://172.30.120.85:9200",
 "username": "elastic",
 "password": "root123!"
 },
 "index": "test_data1"
 },
 "dest": {
 "index": "test_data10"
 }
}
```

2. 执行完成后，命令行中会返回以下数据：

```
{
 "task" : "GxbeiC6NT3apWh6potbpkA:38152"
}
```

3. 查看任务详情：

```
GET /_tasks/GxbeiC6NT3apWh6potbpkA:38152
{
 "completed" : true,
 "task" : {
 "node" : "GxbeiC6NT3apWh6potbpkA",
 "id" : 38152,
 "type" : "transport",
 "action" : "indices:data/write/reindex",
 "status" : {
 "total" : 9000,
 "updated" : 0,
 "created" : 9000,
 }
 }
}
```

```
"deleted" : 0,
"batches" : 9,
"version_conflicts" : 0,
"noops" : 0,
"retries" : {
 "bulk" : 0,
 "search" : 0
},
"throttled_millis" : 0,
"requests_per_second" : -1.0,
"throttled_until_millis" : 0
},
"description" : """reindex from [host=172.30.120.85 port=9200 query={
"match_all" : {
 "boost" : 1.0
}
} username=elastic password=<>>][test_data1] to [test_data10][_doc]""",
"start_time_in_millis" : 1718203864735,
"running_time_in_nanos" : 6411335610,
"cancellable" : true,
"cancelled" : false,
"headers" : { },
},
"response" : {
 "took" : 6399,
 "timed_out" : false,
 "total" : 9000,
 "updated" : 0,
 "created" : 9000,
 "deleted" : 0,
 "batches" : 9,
 "version_conflicts" : 0,
 "noops" : 0,
 "retries" : {
 "bulk" : 0,
 "search" : 0
 },
 "throttled" : "0s",
 "throttled_millis" : 0,
 "requests_per_second" : -1.0,
 "throttled_until" : "0s",
 "throttled_until_millis" : 0,
 "failures" : []
}
```

```

 }
 }
}
```

## 方案建议

建议分多次进行 reindex，并且在 reindex 的时候加上筛选条件，比如指定的 index 有个 last\_updated 字段，第一次 reindex 的时候，可以限制 last\_updated 在某时间点之前：

```
POST _reindex
{
 "source": {
 "remote": {
 "host": "http://172.30.120.85:9200",
 "username": "elastic",
 "password": "root123!"
 },
 "index": "test_data100",
 "query": {
 "range": {
 "last_updated": {
 "lte": 1718265586034
 }
 }
 }
 },
 "dest": {
 "index": "test_data100"
 }
}
```

第二次的时候可以将 range.last\_updated 设置为 {"gt": 1718265586034}。

!!! note

完成 reindex 后，需要验证数据的完整性。

---

## 参考文档

- [Migrating data | Elasticsearch Service Documentation | Elastic](#)
- [Reindex from a remote cluster | Elasticsearch Guide \[7.15\] | Elastic](#)
- [Migrating an Elasticsearch cluster with 0 downtime | by Flavien Berwick | Medium](#)
- [Elasticsearch 跨集群数据迁移方案总结 - 腾讯云开发者社区 - 腾讯云 \(tencent.com\)](#)

# 什么是 Kafka

Kafka 模块是一款基于开源软件 Kafka 提供的分布式消息队列服务。 DaoCloud 为其开发了简单易用的图形化界面，向用户提供计算、存储和带宽资源独占的 Kafka 专享实例。

Kafka 是一个拥有高吞吐、可持久化、可水平扩展，支持流式数据处理等多种特性的分布式消息流处理中间件，采用分布式消息发布与订阅机制，在日志收集、流式数据传输、在线/离线系统分析、实时监控等领域有广泛的应用。

目前支持的特性如下：

- 创建/更新/删除 Kafka 实例
- 可靠性保证机制
- 支持自定义参数配置
- 支持集群高可用
- 多语言客户端支持
- 简单易用的图形界面

kafka 主界面

kafka 主界面

[创建 Kafka 实例](#)

# 使用场景

相比支持广播、事务消息、消息路由、死信队列、优先级队列等且广泛应用于秒杀、流控、系统解耦等场景的 [RabbitMQ](#)，Kafka 消息队列适用于构建实时数据管道、流式数据处理、

第三方解耦、流量削峰去谷等场景，具有大规模、高可靠、高并发访问、可扩展且完全托管的特点。

## 与 RabbitMQ 对比

正所谓没有最好的技术，只有最合适的技术。每个消息中间件服务都有自己的优劣，以下对 RabbitMQ 和 Kafka 做一个简单的对比。

	Kafka	RabbitMQ
性能	单节点 QPS 达百万级，吞吐量高	单节点 QPS 为万级，吞吐量低
可靠性	多副本机制，数据可靠性高	多副本机制，数据可靠性高
功能	持久化事务消息单分区级别的顺序性	持久化优先级队列延迟队列死信队列事务消息
消费模式	消息过滤客户端主动拉取消息回溯广播	客户端主动拉取和服务端推送广播消费
客户端支持	只支持 Kafka 自定义协议采用 Scala 和 Java 编写支持 SSL/SASL 认证和读写权限控制	支持 MQTT、STOMP 等多种协议采用 Erlang 编写支持 SSL/SASL 认证和读写权限控制
服务可用性	采用集群部署，分区与多副本的设计，使用单代理宕机对服务无影响，且支持消息容量的线性提升	支持集群部署，集群代理数量有多种规格
其他	消息堆积流量控制：支持 client 和 user 级别，通过主动设置可将流控作用于生产者或消费者	消息追踪消息堆积多租户流量控制：流控基于 Credit-based 算法，是内部被动触发的保护机制，作用于生产者层面

总之，Kafka 采用拉取（Pull）方式消费消息，吞吐量相对更高，适用于海量数据收集与传递场景，例如日志采集和集中分析。

而 RabbitMQ 基于 Erlang 语言开发，不利于做二次开发和维护，适用于对路由、负载均衡、数据一致性、稳定性和可靠性要求很高，对性能和吞吐量要求没那么高的场景。

## 典型场景

Kafka 作为一款热门的消息队列中间件，具备高效可靠的消息异步传递机制，主要用于不同系统间的数据交流和传递，在企业解决方案、金融支付、电信、电子商务、社交、即时通信、视频、物联网、车联网等众多领域都有广泛应用。

### 1. 异步通信

将业务中属于非核心或不重要的流程部分使用消息异步通知的方式发给目标系统，这样主业务流程无需同步等待其他系统的处理结果，从而达到系统快速响应的目的。如网站的用户注册场景，在用户注册成功后，还需要发送注册邮件与注册短信，这两个流程使用 Kafka 消息服务通知邮件发送系统与短信发送系统，从而提升注册流程的响应速度。

### 2. 错峰流控与流量削峰

在电子商务或大型网站中，上下游系统处理能力存在差异，处理能力高的上游系统的突发流量可能会对处理能力低的某些下游系统造成冲击，需要提高系统的可用性的同时降低系统实现的复杂性。电商大促销等流量洪流突然来袭时，可以通过队列服务堆积缓存订单等信息，在下游系统有能力处理消息的时候再处理，避免下游订阅系统因突发流量崩溃。消息队列提供亿级消息堆积能力，3 天的默认保留时长，消息消费系统可以错峰进行消息处理。

另外，在商品秒杀、抢购等流量短时间内暴增场景中，为了防止后端应用被压垮，可

在前后端系统间使用 Kafka 消息队列传递请求。

### 3. 日志同步

在大型业务系统设计中，为了快速定位问题，全链路追踪日志，以及故障及时预警监

控，通常需要将各系统应用的日志集中分析处理。

Kafka 设计初衷就是为了应对大量日志传输场景，应用通过可靠异步方式将日志消息

同步到消息服务，再通过其他组件对日志做实时或离线分析，也可用于关键日志信

息收集进行应用监控。

日志同步主要有三个关键部分：日志采集客户端，Kafka 消息队列以及后端的日志处

理应用。

## 离线升级中间件 - kafka 模块

本页说明从[下载中心](#)下载中间件 - kafka 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 \_\_mcamel\_\_ 字样是中间件模块的内部开发代号。

### 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer

同步镜像到镜像仓库，该方法更加高效便捷。

#### chart-syncer 同步镜像到镜像仓库

##### 1. 创建 load-image.yaml

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

#### ==== “已安装 chart repo”

若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
  url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

#### ==== “未安装 chart repo”

若当前环境未安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: LOCAL
  path: ./local-repo # chart 本地路径
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
tar -xvf mcamel-kafka_0.8.1_amd64.tar
cd mccamel-kafka_0.8.1_amd64
tar -xvf mccamel-kafka_0.8.1.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

### !!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

### ==== “通过 helm repo 升级”

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep kafka
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-kafka http://{harbor url}/chartrepo/{project}
```

```

1. 更新 helm 仓库。

```
```shell
helm repo update mcamel/mcamel-kafka # helm 版本过低会导致失败，若失败，请尝试执行 helm update repo
```

```

1. 选择您想安装的版本（建议安装最新版本）。

```
```shell
helm search repo mcamel/mcamel-kafka --versions
```

```none
[root@master ~]# helm search repo mcamel/mcamel-kafka --versions
NAME          CHART VERSION   APP VERSION   DESCRIPTION
mcamel/mcamel-kafka    0.8.1        0.8.1        A Helm chart for Kubernetes
...
```

```

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-kafka -n mcamel-system -o yaml > mcamel-kafka.yaml
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 `mcamel-kafka.yaml` 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-kafka mcamel/mcamel-kafka \
-n mcamel-system \
-f ./mcamel-kafka.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.8.1
```

```
```

==== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-kafka -n mcamel-system -o yaml > mcamel-kafka.yaml
```

```
```

1. 执行 `helm upgrade` 。

升级前建议您覆盖 `bak.yaml` 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-kafka . \
-n mcamel-system \
-f ./mcamel-kafka.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```

```
```

Kafka 常见术语

- 消息和批次

Kafka 的基本数据单元称为 message (消息) , 为减少网络开销 , 提高效率 , 多个消息会被放入同一批次 (Batch) 中后再写入。

- 主题和分区

Kafka 的消息通过 Topic (主题) 进行分类 , 一个主题可以分为若干个 Partition (分区) , 一个分区就是一个提交日志 (commit log)。消息以追加的方式写入分区 , 然后以先入先出的顺序读取。Kafka 通过分区来实现数据的冗余和伸缩性 , 分区可以分布在不同的服务器上 , 这意味着一个 Topic 可以横跨多个服务器 , 以提供比单个服务器更强大的性能。

由于一个 Topic 包含多个分区 , 因此无法在整个 Topic 范围内保证消息的顺序性 , 但可以保证消息在单个分区内的顺序性。

主题和分区

主题和分区

- 生产者和消费者

生产者负责创建消息。一般情况下 , 生产者在把消息均衡地分布到在主题的所有分区上 , 而并不关心消息会被写到哪个分区。如果我们想要把消息写到指定的分区 , 可以通过自定义分区器来实现。

消费者是消费者群组的一部分 , 消费者负责消费消息。消费者可以订阅一个或者多个主题 , 并按照消息生成的顺序来读取它们。消费者通过检查消息的偏移量 (offset) 来区分读取过的消息。偏移量是一个不断递增的数值 , 在创建消息时 , Kafka 会

把它添加到其中，在给定的分区里，每个消息的偏移量都是唯一的。消费者把每个分区最后读取的偏移量保存在 Zookeeper 或 Kafka 上，如果消费者关闭或者重启，它还可以重新获取该偏移量，以保证读取状态不会丢失。

消费者

消费者

一个分区只能被同一个群组中的一个消费者读取，但可以被不同群组中的多个消费者共同读取。多个群组中的消费者共同读取同一个主题时，彼此之间互不影响。

消费者

消费者

- Broker 和 Cluster

一个独立的 Kafka 服务器被称为 Broker。Broker 接收来自生产者的消息，为消息设置偏移量，并提交消息到磁盘保存。Broker 为消费者提供服务，对读取分区的请求做出响应，返回已经提交到磁盘的消息。

Broker 是集群 (Cluster) 的组成部分。每个集群都会选举出一个 Broker 作为集群控制器 (Controller)，集群控制器负责管理工作，包括将分区分配给 Broker 和监控 Broker。

在集群中，一个分区 (Partition) 从属一个 Broker，该 Broker 被称为分区的首领 (Leader)。一个分区可以分配给多个 Broker，这个时候会发生分区复制。这种复制机制为分区提供了消息冗余，如果有一个 Broker 失效，其他 Broker 可以接管领导权。

broker 和 cluster

broker 和 cluster

Kafka 消息队列 Release Notes

本页列出 Kafka 消息队列的 Release Notes，便于您了解各版本的演进路径和特性变化

*[mcamel-kafka]: mcamel 是 DaoCloud 所有中间件的开发代号，kafka

2025-02-28

v0.23.0

- 优化 将 Kafka 管理工具由 CMAK 更新为 kafka-ui

2024-11-30

v0.21.0

- 优化 升级 kafka-operator 镜像版本到 0.40.0，并支持创建 3.7.0 版本的 kafka 实例
- 优化 增加 kafka-operator 的内存限制量

2024-09-30

v0.19.0

- 修复 选择工作空间查询 Kafka 列表时权限泄漏的问题
- 修复 部分操作无审计日志的问题

2024-08-31**v0.18.0**

- **优化** 创建实例时不可选择异常的集群

2024-05-31**v0.15.0**

- **新增** 参数模板导入功能
- **优化** 支持批量修改实例参数
- **优化** 支持单独配置 zookeeper 的亲和性
- **修复** 容器组列表里 Manager 资源无法正常展示

2024-04-30**v0.14.0**

- **优化** 增加命名空间配额的提示
- **修复** 参数模板默认值不正确的问题

2024-03-31**v0.13.0**

- **新增** 支持参数模板管理
- **新增** 支持通过模板创建 Kafka 实例

- 优化 支持为 zookeeper 配置存储
- 优化 当用户权限不足时无法读取 kafka 的密码

2024-01-31

v0.12.0

- 优化 在全局管理中增加 Kafka 版本展示

2023-12-31

v0.11.0

- 优化 监控面板支持中文
- 优化 Kafka 采用 NodePort 模式访问时返回多个 broker 的 NodePort
- 修复 Kafka 修复列表展示重复数据的问题
- 修复 创建实例时部分输入框填写特殊字符的校验未生效的问题

2023-11-30

v0.10.0

- 新增 支持记录操作审计日志
- 优化 实例列表未获取到列表信息时的提示

2023-10-31**v0.9.0**

- **新增** 离线升级
- **新增** 实例重启功能
- **修复** cloudshell 权限问题

2023-08-31**v0.8.0**

- **新增** 白名单访问控制
- **优化** KindBase 语法兼容
- **优化** operator 创建过程的页面展示

2023-07-31**v0.7.3**

- **新增** UI 界面的权限访问限制

2023-06-30**v0.7.0**

- **新增** 对接全局管理审计日志模块
- **新增** LoadBalancer 服务类型

- 优化 监控图表，去除干扰元素并新增时间范围选择
- 优化 **ServiceMonitor** 闭环安装

2023-04-27

v0.5.1

- 新增 **mcamel-kafka** 详情页面展示相关的事件
- 新增 **mcamel-kafka** 支持自定义角色
- 优化 **mcamel-kafka** 调度策略增加滑动按钮

2023-03-28

v0.4.0

- 新增 **mcamel-kafka** 支持中间件链路追踪适配
- 新增 安装 **mcamel-kafka** 根据参数配置启用链路追踪
- 优化 **mcamel-kafka** 优化 Kafka 的默认配置
- 升级 golang.org/x/net 到 v0.7.0
- 升级 GHippo SDK 到 v0.14.0

2023-02-23

v0.3.0

新功能

- 新增 **mcamel-kafka** helm-docs 模板文件

- 新增 **mcamel-kafka** 应用商店中的 Operator 只能安装在 mcamel-system
- 新增 **mcamel-kafka** 支持 cloud shell
- 新增 **mcamel-kafka** 支持导航栏单独注册
- 新增 **mcamel-kafka** 支持查看日志
- 新增 **mcamel-kafka** Operator 对接 chart-syncer

优化

- 优化 **mcamel-kafka** 升级离线镜像检测脚本

修复

- 修复 **mcamel-kafka** 实例名太长导致自定义资源无法创建的问题
- 修复 **mcamel-kafka** 工作空间 Editor 用户无法查看实例密码
- 新增 日志查看操作说明，支持自定义查询、导出等功能

2022-12-25

v0.2.0

- 新增 **mcamel-kafka** NodePort 端口冲突提前检测
- 新增 **mcamel-kafka** 节点亲和性配置
- 优化 **mcamel-kafka** manager 去掉 probe，防止 kafka 没准备好不能打开 manager
- 优化 **mcamel-kafka** zooEntrance 重新打包镜像地址为 1.0.0

2022-11-28**v0.1.6**

- **改进** 完善优化复制功能
- **改进** 实例详情 - 访问设置，移除集群 IPv4
- **改进** 中间件密码校验难度调整
- **新增** 对接告警能力
- **新增** 新增判断 sc 是否支持扩容并提前提示功能
- **优化** 优化安装环境检测的提示逻辑 & 调整其样式
- **优化** 中间件样式走查优化
- **修复** 离线镜像有数字和大写无法被扫描到

2022-11-08**v0.1.4**

- **修复** 更新时无法校验到正确字段，如 managerPass
- **改进** 密码校验调整为 MCamel 低等密码强度
- **新增** 返回是否可以更新 sc 容量的校验
- **新增** 返回列表或者详情时的公共字段
- **新增** 返回告警
- **新增** 校验 Service 注释
- **修复** operator 用名字选择
- **修复** 服务地址展示错误

- **修复** Kafka 使用 NodePort 时，创建失败

2022-10-28

v0.1.2

- **新增** 同步 Pod 状态到实例详情页
- **优化** workspace 界面逻辑调整
- **优化** 不符合设计规范的样式调整
- **优化** password 获取逻辑调整
- **优化** cpu&内存请求量应该小于限制量逻辑调整

2022-9-25

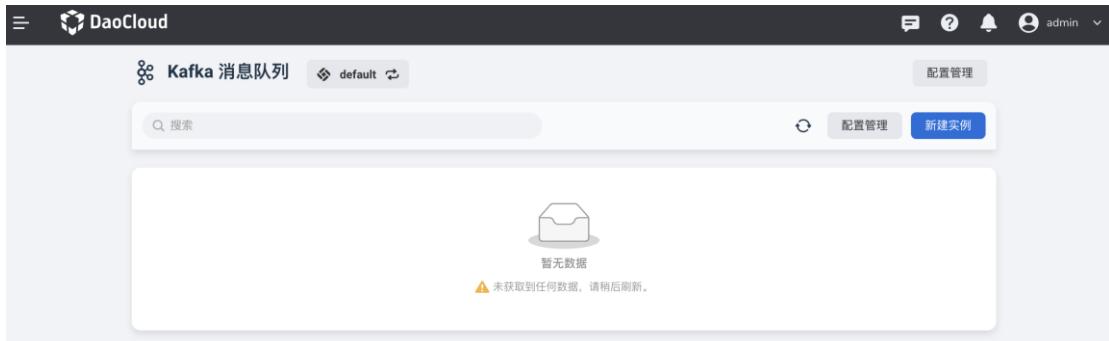
v0.1.1

- **新增** 支持 kafka 列表查询，状态查询，创建，删除和修改
- **新增** 支持 kafka-manager 对 kafka 进行管理
- **新增** 支持 kafka 的指标监控，查看监控图表
- **新增** 支持 ghippo 权限联动
- **新增** mcamel-kafka 获取用户列表接口
- **优化** 更新 release note 脚本，执行 release-process 规范

创建 Kafka

在 Kafka 消息队列中，执行以下操作创建 Kafka 实例。

1. 在 Kafka 消息队列页面，点击右上角的 **新建实例** 按钮。



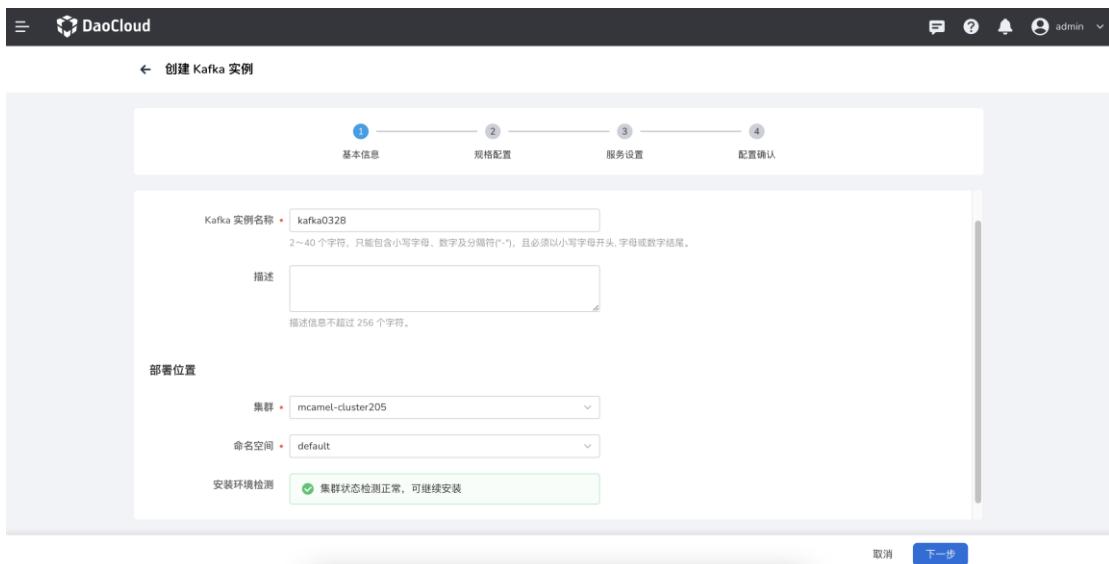
新建实例

!!! tip

初次部署时，可以点击 ****立即部署****。

![新建实例](docs/zh/docs/middleware/kafka/images/create011.png)

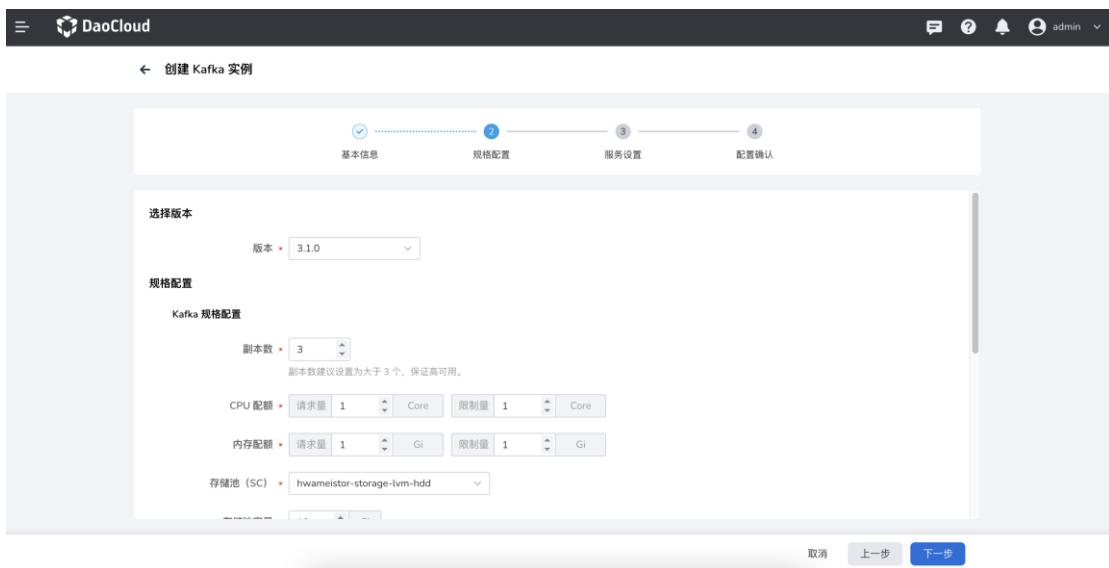
2. 在 **创建 Kafka 实例** 页面中，设置基本信息后，点击 **下一步**。



设置基本信息

3. 配置规格后，点击 **下一步**。

- 版本：Kafka 的版本号，当前仅支持 Kafka 3.1.0。
- 副本数：支持 1、3、5、7 副本数。
- 资源配额：根据实际情况选择规则。
- 存储卷：选择 Kafka 实例的存储卷和储存空间总量。



配置规格

4. 服务设置后，点击 **下一步**。

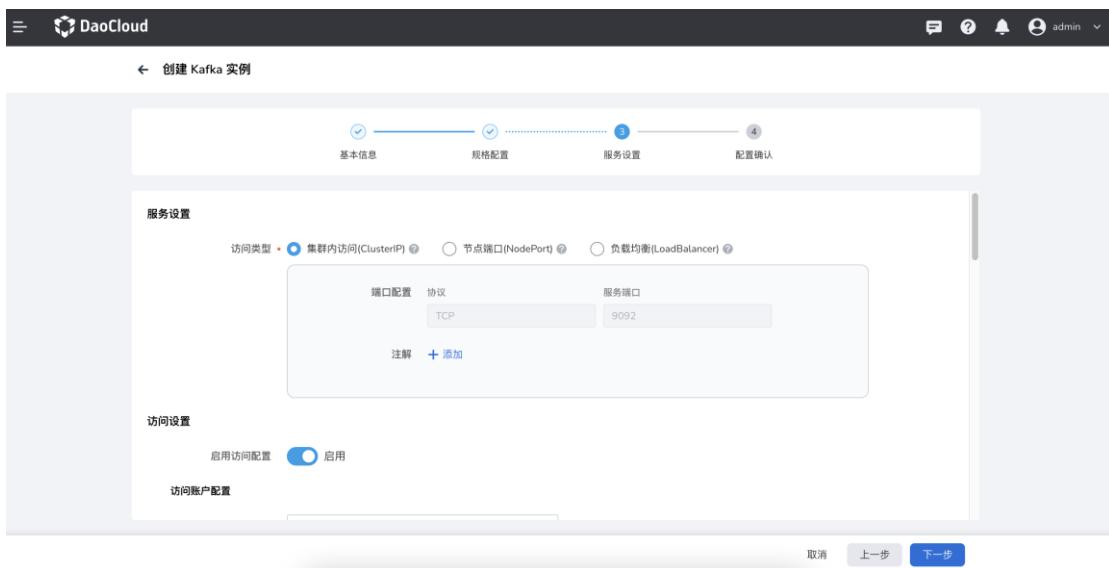
- 服务设置：

- 集群内访问 (ClusterIP)
- 节点端口 (Nodeport)
- 负载均衡 (LoadBalancer)

- 访问设置：

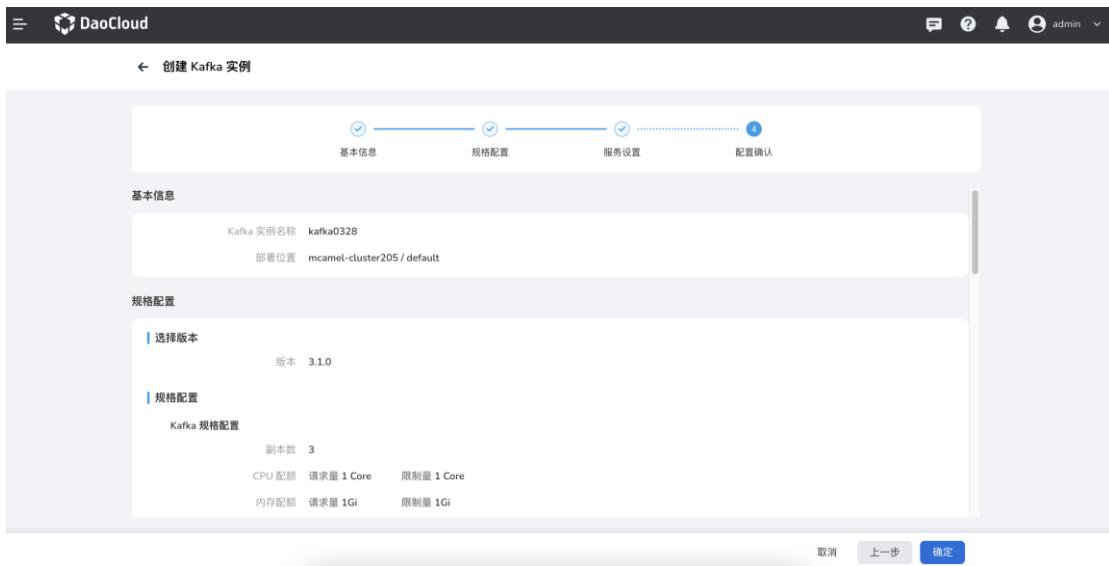
- 访问账户配置：连接 Kafka 实例的用户名、密码
- CMAK 资源配置：副本数、CPU 和内存配额
- 访问类型配置：节点端口 (Nodeport)、负载均衡 (LoadBalancer)

- 高级设置：按需配置



服务设置

5. 确认实例配置信息无误，点击 **确定** 完成创建。



点击确定

6. 在实例列表页查看实例是否创建成功。刚创建的实例状态为 **未就绪**，等几分钟后该状态变为 **运行中**。

The screenshot shows the Kafka message queue management interface. At the top, there's a search bar and a '配置管理' (Configuration Management) button. Below the search bar, the Kafka instance 'test0328' is listed with a green status indicator. The instance details include: 部署位置 (Deployment Location) - kpanda-global-cluster/mca...; Kafka 版本 (Version) - 3.1.0; 访问地址 (Access Address) - test0328-kafka-service.mca...; 资源配额 (Resource Quota) - CPU 请求量 1 Core, 内存 请求量 1 GB, 磁盘 10 GB; Kafka 副本数 (Kafka Replicas) - 3 / 3; Zookeeper 副本数 (Zookeeper Replicas) - 3 / 3. At the bottom, there's a pagination bar showing 1 / 1 and 10 项.

查看状态

!!! note

另外 DCE 5.0 的 Kafka 提供了参数模板来简化实例的创建。
您可以使用这些预设的[参数模板](./template.md)来创建实例。

查看、更新和删除 Kafka

本节说明如何查看 Kafka 消息队列，并执行后续的更新及删除操作。

查看 Kafka 消息队列

1. 在消息队列页面，点击某个实例名称。

The screenshot shows the Kafka message queue management interface. A specific Kafka instance named 'kebe-test' is highlighted with a red circle and labeled '未就绪' (Not Ready). The instance details are: 部署位置 (Deployment Location) - gpu-cluster/demo; Kafka 版本 (Version) - 3.1.0; 访问地址 (Access Address) - kebe-test-kafka-service.demo...; 资源配额 (Resource Quota) - CPU 请求量 1 Core, 内存 请求量 1 Gi, 磁盘 10 Gi; Kafka 副本数 (Kafka Replicas) - 1 / 1; Zookeeper 副本数 (Zookeeper Replicas) - 1 / 1. At the bottom, there's a pagination bar showing 1 / 1 and 10 项.

点击实例

2. 进入消息队列概览，查看基本信息、访问设置、资源配额和 Pod 列表等信息。

The screenshot shows the Kafka instance configuration page. Key details include:

- 基本信息:** Instance name: kebe-test, Status: 未就绪 (Not Ready), Location: gpu-cluster/demo, Created time: 2024-06-03 11:18, Version: 3.1.0.
- 访问设置 (Access Settings):** NodePort: http://10.20.100.213:32238, CMAK 访问地址: kafka*****.
- 资源配额 (Resource Quotas):** CPU: 1 Core, 内存: 1 Gi, 磁盘: 10 Gi.
- 监控告警 (Monitoring Alerts):** No alerts present.
- 容器组列表 (Container Group List):** Shows two container groups: kebe-test-zookeeper and kebe-test-manage.

查看信息

更新 Kafka 消息队列

1. 在消息队列中，点击右侧的 ... 按钮，在弹出菜单中选择 **更新实例**。

The screenshot shows the Kafka message queue instance list. A context menu is open over the 'test-middleware' instance, showing options: 更新实例 (Update Instance), 更副本数 (Change Replicas), 访问名重置 (Reset Access Name), 重启 (Restart), and 停止实例 (Stop Instance). The '更新实例' option is highlighted with a red box.

选择更新实例

2. 修改基本信息、规格配置及服务设置后，点击 **确定**。

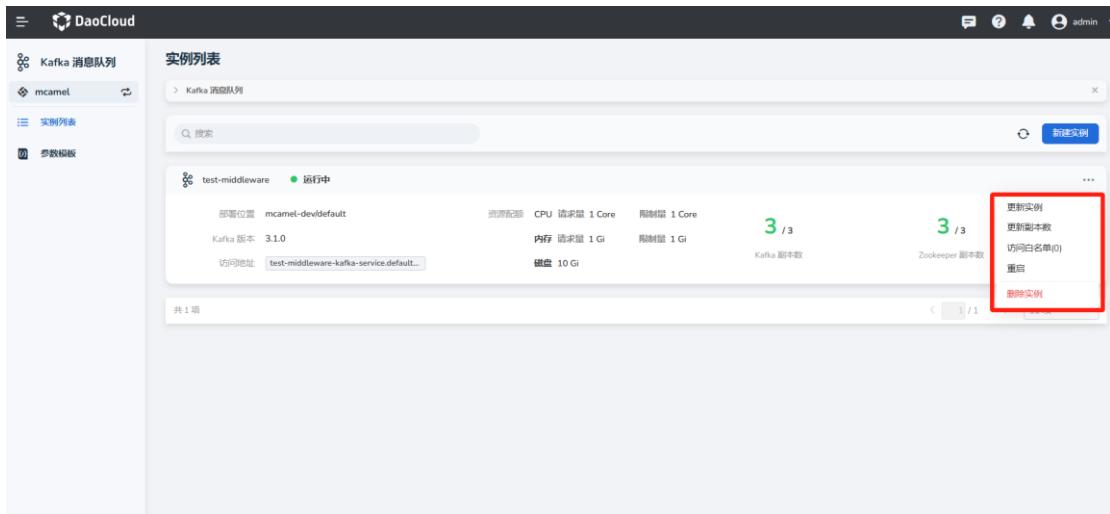
服务设置

服务设置

3. 返回消息队列，屏幕右上角将显示消息：**更新实例成功**。

删除 Kafka 消息队列

1. 选择一个消息队列，点击右侧的 ... 按钮，在弹出菜单中选择 **删除实例**。



选择删除实例

2. 在弹窗中输入该消息队列的名称，确认无误后，点击 **删除** 按钮。

点击删除

点击删除

!!! warning

删除实例后，该实例相关的所有消息也会被全部删除，请谨慎操作。

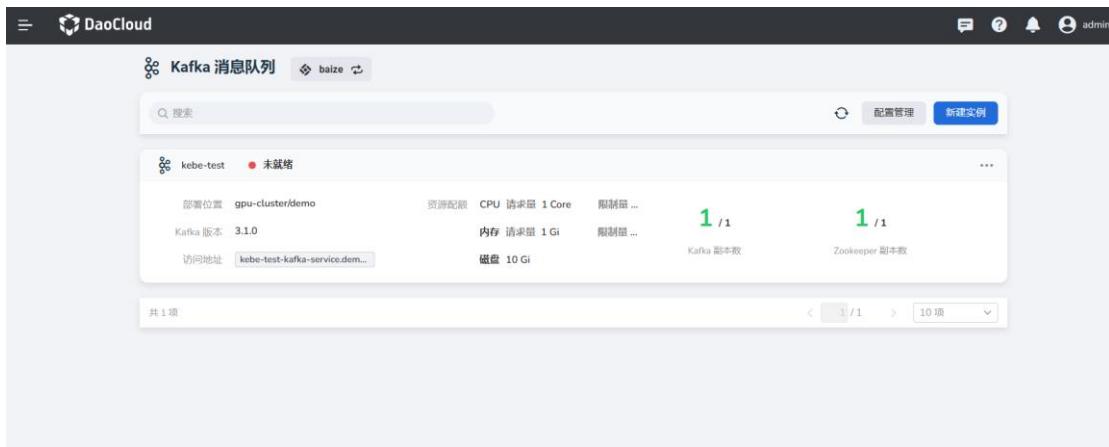
实例监控与日志查看

本节说明如何进行 Kafka 实例监控并查看日志。

实例监控

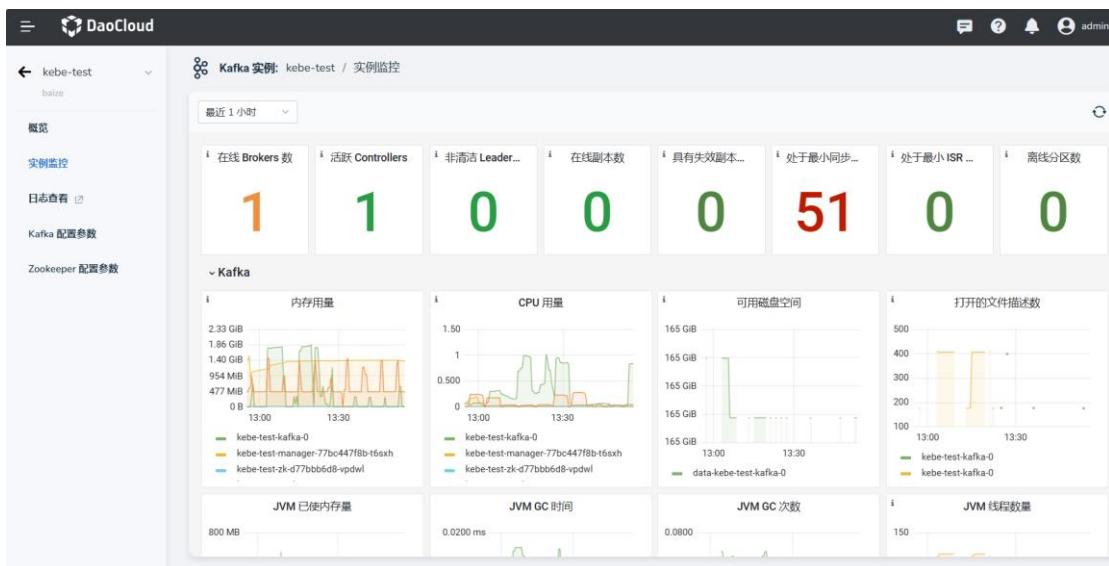
Kafka 内置了 Prometheus 和 Grafana 监控模块。

1. 在消息队列页面，点击某个实例名称进入到实例详情页面。



点击实例

2. 在左侧导航栏，点击 实例监控，可以接入监控模块。



点击实例监控

日志查看

通过访问每个 Kafka 的实例详情页面，可以查看 Kafka 的日志。

1. 在消息队列页面，选择想要查看的实例，点击名称进入到实例详情页面。

2. 在左侧导航栏，点击 日志查看 即可进入到日志查看页面 ([Insight 日志查看](#))。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 自定义日志时间范围，在日志页面右上角，可以方便地切换查看日志的时间范围
(可查看的日志范围以 可观测系统设置内保存的日志时长为准)
- 支持 关键字检索日志，左侧检索区域支持查看更多的日志信息
- 支持 日志量分布查看，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 查看日志的上下文，点击右侧 **上下文** 图标即可
- 支持 导出日志

image

image

配置参数

Kafka 内置了参数配置 UI 界面。

1. 在消息队列页面中，点击某个名称。

点击某个名称

点击某个名称

2. 在左侧导航栏，点击 **配置参数**。

点击配置参数

点击配置参数

3. 点击 **编辑配置** 滑块开关，可以很方便地配置 Kafka 和 Zookeeper 的各项参数。

配置参数

配置参数

4. 点击 **保存**，参数将立即生效，不会发生重启。

参数模板

DCE 5.0 Kafka 提供了参数模板功能，方便实例的创建。

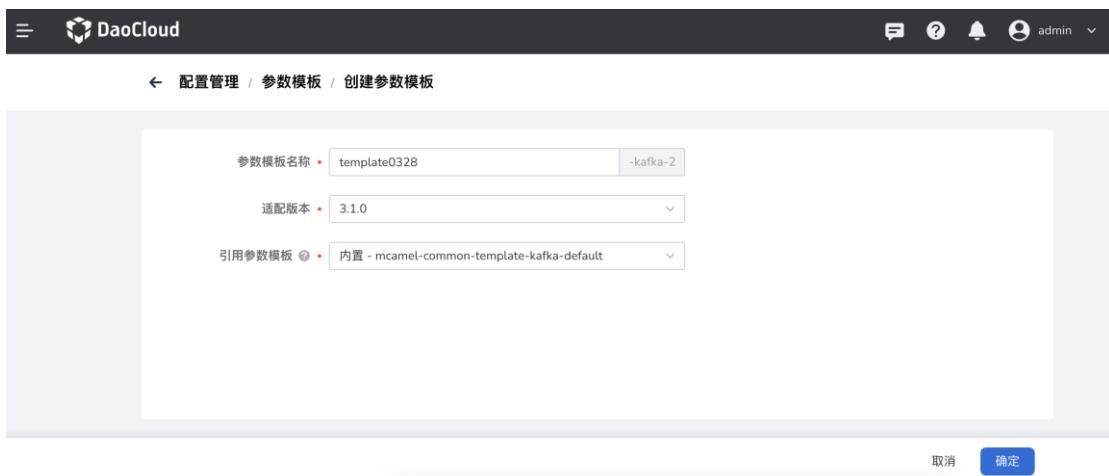
1. 在 Kafka 消息队列页面，点击右上角的 **配置管理**。

点击配置管理

2. 点击 **创建模板** 按钮。

点击创建模板

3. 输入名称，选择适配的版本，引用某个内置的参数模板后，点击 **确定**。



填写参数

4. 屏幕提示创建成功，刷新页面，点击右侧的⋮，可以执行更多操作。



更多操作

下一步：[创建 Kafka 实例](#)

Kafka 双机房部署灾备方案

背景

在生产上线时，为了保证业务的持续不中断，通常使用双机房的部署方案达到一个机房故障时，另一个机房中的应用仍然能够继续提供服务。对于 Kafka 而言，在双机房的部署下，希望一个 partition 的副本尽可能的处在不同的机房。此处 Kafka 仍然部署在同一个集群

中，集群中的节点处于不同的机房。

Rack awareness (机架感知) 功能介绍

Rack awareness 特性会将同一 partition 的 replica 分散到不同的 rack 上。该特性扩展了 kafka 为 broker-failure 所提供的保证，使其可以覆盖 rack-failure，减少了同一 rack 上所有 broker 同时失败会导致的数据丢失的风险。该特性也可以应用到其他 broker grouping 上，例如 aws 上的 EC2 可用区，可以通过 broker.rack 指定某个 broker 属于某个特定的 rack。

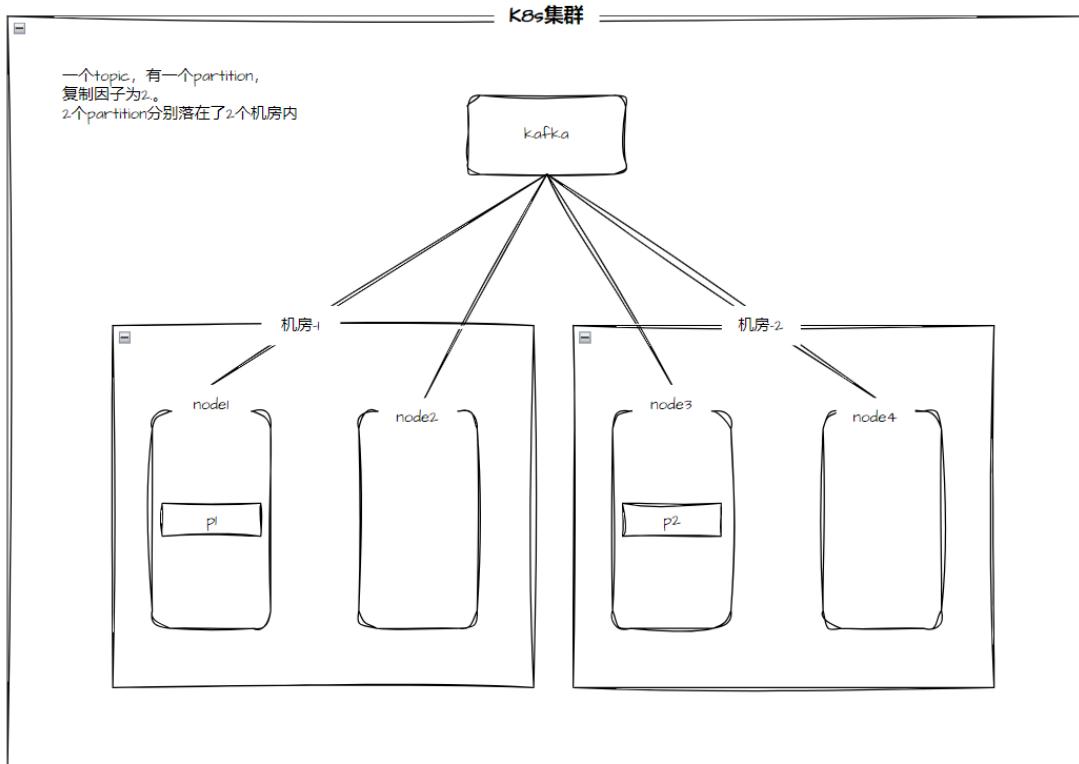
当新建/修改 topic 或重分布 replica 时，kafka 会考虑 broker.rack 配置，确保 replica 分散到尽量多的 rack 上，一定注意，是尽量在 rack 间，而不是 broker 间，平分 replica。一个 partition 将横跨 $\min(racks, replication-factor)$ 个不同的 rack。

- 不配置 broker.rack 时，分配 replica 到 broker 的算法确保每个 broker 上 leader 的个数 (partition 的 leader 的个数) 是相同的，而不论 broker 在 racks 间如何分布。这确保了 balanced throughput，即各个 broker 的吞吐是一样的。
- 配置 broker.rack 时，需要在 rack 之间平分所有 replica，也就是“不管在 rack 上分配了几个 broker，反正 replica 是按着 rack 无脑平分的”。正因为此，如果 rack 被分配了不同个数的 broker，则较少 broker 的 rack (其 replica 并不少) 将使用更多的 storage，会把更多的资源用到 replication 上，这些 broker 的负担就会更重，当然负载也不均衡，所以，明智的做法是为每个 rack 配置同样数量的 broker。

举个例子，假设 2 个 rack，6 个 broker，100 个 replica (包括主和从)，replica 会在 rack 间无脑平分，因此，rack1 和 rack2 各自分配 50 个 replica：

1. 如果 rack1 和 rack2 都分配 3 个 broker，则：rack1 上的每一个 broker 分配 50/3 个 replica，rack2 上的每一个 broker 分配 50/3 个 replica。
2. 如果 rack1 分配 5 个 broker，rack2 分配 1 个 broker，则：rack1 上的每一个 broker 分配 50/5 个 replica，rack2 上的每一个 broker 分配 50/1 个 replica。也就是：具有较少 broker 的 rack2 其 replica 一点都不少，所以这些 broker 负担会更重，当然负载也不均衡。所以，明智的做法是为每个 rack 配置同样数量的 broker。

双机房部署架构



kafka

操作步骤

1. 登录目标集群的控制台，执行以下操作，分别为在不同机房的的节点打标签（Label）。

```
kubectl label node master01 topology.kubernetes.io/zone=east #这里的 east 可以换成自定  
义的机房名称
```

```
kubectl label node worker01 topology.kubernetes.io/zone=west #这里的 east 可以换成  
自定义的机房名称
```

2. 修改 Kafka CR，在 CR 资源上配置 rack 特性。

```
apiVersion: kafka.strimzi.io/v1beta2  
kind: Kafka  
metadata:  
  name: test-rack  
spec:  
  kafka:  
    rack:  
      topologyKey: topology.kubernetes.io/zone # 这里的 value 应该为第一步中所打  
label 的 key  
      version: 3.1.0  
      replicas: 4  
      listeners:  
        - name: plain  
          port: 9092  
          type: internal  
          tls: false  
        - name: tls  
          port: 9093  
          type: internal  
          tls: true  
      config:  
        offsets.topic.replication.factor: 3  
        transaction.state.log.replication.factor: 3  
        transaction.state.log.min_isr: 2  
        default.replication.factor: 3  
        min.insync.replicas: 2  
        inter.broker.protocol.version: "3.1"  
      storage:  
        class: local-path  
        size: 1Gi  
        type: persistent-claim  
    zookeeper:  
      replicas: 1  
      storage:  
        class: local-path  
        size: 1Gi  
        type: persistent-claim
```

```
entityOperator:
  topicOperator: {}
  userOperator: {}
```

- 部署完成后，Kafka 对应容器组的调度分布如下所示：

test-rack-kafka-0			1/1	Running	0	89m
10.244.2.117	worker01	<none>		<none>		
test-rack-kafka-1			1/1	Running	0	89
m 10.244.0.109	master01	<none>		<none>		
test-rack-kafka-2			1/1	Running	0	89
m 10.244.2.118	worker01	<none>		<none>		
test-rack-kafka-3			1/1	Running	0	12
m 10.244.0.111	master01	<none>		<none>		

- kafka-operator 会自动为每一个 kafka 的容器组添加

`preferredDuringSchedulingIgnoredDuringExecution` 亲和性配置，如下代码所示：

`podAntiAffinity:`

`preferredDuringSchedulingIgnoredDuringExecution:`

`- weight: 100`

`podAffinityTerm:`

`labelSelector:`

`matchLabels:`

`strimzi.io/cluster: test-rack`

`strimzi.io/name: test-rack-kafka`

`topologyKey: topology.kubernetes.io/zone`

- 进入 kafka 容器组检查内部的配置文件同样自动添加 `broker.rack` 配置，如下代码所示：

```
[kafka@test-rack-kafka-2 custom-config]$ cat server.config
#####
#####
# This file is automatically generated by the Strimzi Cluster Operator
# Any changes to this file will be ignored and overwritten!
#####
#####

#####
# Broker ID
#####
broker.id=2
node.id=2
```

```
#####
# Rack ID
#####
broker.rack=${STRIMZI_RACK_ID} // cat /opt/kafka/init/rack.id
```

测试用例

- -- topic 下 1 个 partition, 复制因子为 4, 均匀分布在每个 kafka 的节点上
`./bin/kafka-topics.sh --create --bootstrap-server localhost: 9092 --replication-factor 4 --partitions 1 --topic test-1`

```
[kafka@test-rack-kafka-1 kafka]$ ./bin/kafka-topics.sh --describe --bootstrap-server localhost: 9092 --topic test-1
Topic: test-1 TopicId: -CNSnHsCT9eG7znhr5CYxA PartitionCount: 1 ReplicationFactor: 4
Configs: min.insync.replicas=2,message.format.version=3.0-IV1
Topic: test-1 Partition: 0 Leader: 1 Replicas: 1,0,3,2 Isr: 1,0,3,2
```

- --topic 下 1 个 partition, 复制因子为 2, 分别分布在 west(0) 和 east(3) 上, "机房" 上的分布是均匀的
`./bin/kafka-topics.sh --create --bootstrap-server localhost: 9092 --replication-factor 2 --partitions 1 --topic test-1`

```
[kafka@test-rack-kafka-1 kafka]$ ./bin/kafka-topics.sh --describe --bootstrap-server localhost: 9092 --topic test-2
Topic: test-2 TopicId: u_KNAaLjT3atQy_DTkmZfQ PartitionCount: 1 ReplicationFactor: 2
Configs: min.insync.replicas=2,message.format.version=3.0-IV1
Topic: test-2 Partition: 0 Leader: 0 Replicas: 0,3 Isr: 0,3
```

- --topic 下 2 个 partition, 复制因子为 3, "机房" 上的分布不是均匀的, 保证每个 "机房" 都有一个
`./bin/kafka-topics.sh --create --bootstrap-server localhost: 9092 --replication-factor 3 --partitions 2 --topic test-3`

```
[kafka@test-rack-kafka-1 kafka]$ ./bin/kafka-topics.sh --describe --bootstrap-server localhost: 9092 --topic test-3
Topic: test-3 TopicId: qBlTqRh6RCGS4vSukegMzg PartitionCount: 2 ReplicationFactor: 3
Configs: min.insync.replicas=2,message.format.version=3.0-IV1
Topic: test-3 Partition: 0 Leader: 2 Replicas: 2,3,1 Isr: 2,3,1
Topic: test-3 Partition: 1 Leader: 1 Replicas: 1,2,0 Isr: 1,2,0
```

- --topic 下 10 个 partition, 复制因子为 4, "机房" 上的分布是均匀的
`./bin/kafka-topics.sh --create --bootstrap-server localhost: 9092 --replication-factor 4 --partitions 10 --topic test-4`

```
[kafka@test-rack-kafka-2 kafka]$ ./bin/kafka-topics.sh --describe --bootstrap-server localhost: 9092 --topic test-4
```

```

92 --topic test-4
Topic: test-4    TopicId: GDEe60DNST61gGCQX4OhLw PartitionCount: 10      ReplicationFac
tor: 4    Configs: min.insync.replicas=2,message.format.version=3.0-IV1
    Topic: test-4    Partition: 0    Leader: 1    Replicas: 1,2,0,3    Isr: 1,2,0,3
    Topic: test-4    Partition: 1    Leader: 0    Replicas: 0,1,3,2    Isr: 0,1,3,2
    Topic: test-4    Partition: 2    Leader: 3    Replicas: 3,0,2,1    Isr: 3,0,2,1
    Topic: test-4    Partition: 3    Leader: 2    Replicas: 2,3,1,0    Isr: 2,3,1,0
    Topic: test-4    Partition: 4    Leader: 1    Replicas: 1,0,3,2    Isr: 1,0,3,2
    Topic: test-4    Partition: 5    Leader: 0    Replicas: 0,3,2,1    Isr: 0,3,2,1
    Topic: test-4    Partition: 6    Leader: 3    Replicas: 3,2,1,0    Isr: 3,2,1,0
    Topic: test-4    Partition: 7    Leader: 2    Replicas: 2,1,0,3    Isr: 2,1,0,3
    Topic: test-4    Partition: 8    Leader: 1    Replicas: 1,2,0,3    Isr: 1,2,0,3
    Topic: test-4    Partition: 9    Leader: 0    Replicas: 0,1,3,2    Isr: 0,1,3,2

```

什么是 RabbitMQ

RabbitMQ 是实现了高级消息队列协议 (AMQP) 的开源消息代理软件 (亦称面向消息的中间件)，具有高吞吐、低延迟、可扩容等特性。

RabbitMQ 是应用层协议的一个开放标准，为面向消息的中间件设计，基于此协议的客户端与消息中间件来传递消息，不受产品、开发语言等条件的限制。这是一个企业级的真正具备低延迟、高并发、高可用、高可靠特性，可支撑万亿级数据洪峰的分布式消息中间件服务。

rmq 主界面

rmq 主界面

其系统架构及数据流如下图。

架构数据流

架构数据流

[创建 RabbitMQ 实例](#)

功能特性

RabbitMQ 的通用功能特性包括：

- 可靠性 (Reliability)

RabbitMQ 使用一些机制来保证可靠性，如持久化、传输确认、发布确认。

- 消息集群 (Clustering)

多个 RabbitMQ 服务器可以组成一个集群，形成一个逻辑 Broker。

- 高可用队列 (Highly Available Queues)

队列可以在集群中的主机上进行镜像，使得在部分节点出问题的情况下队列仍然可用。

- 多种协议 (Multi-protocol)

RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT 等。

- 多语言客户端 (Many Clients)

RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、Ruby 等。

- 管理界面 (Management UI)

RabbitMQ 提供了一个易用的图形用户界面，使得用户可以监控和管理消息 Broker 的方方面面。

- 跟踪机制 (Tracing)

如果消息异常，RabbitMQ 提供了消息跟踪机制，用户可以轻松找出发生了什么。

- 插件机制 (Plugin System)

RabbitMQ 提供了许多插件，支持从多方面进行扩展，也可以编写自己的插件。

在 DCE 5.0 中部署 RabbitMQ 后，还将支持以下特性：

- 支持单节点和多节点 RabbitMQ 集群部署
- 支持 RabbitMQ Management 插件，提供管理页面
- 支持 RabbitMQ Prometheus 插件，暴露监控指标
- 使用 ServiceMonitor 对接 Prometheus 抓取指标
- 支持 RabbitMQ 集群的扩容和滚动升级

产品优势

RabbitMQ 具有以下产品优势。

- 行业领先
 - 支持 ActiveMQ 的快速部署与生命周期管理。ActiveMQ 是目前最流行的开源多协议消息中间件，支持 JavaScript, C, C++, Python, .Net 等语言编写的应用和平台，能够跨应用交换消息。通过消息控制台实现对生产集群、消费集群、消息查询等的管理操作。
 - 采用业界先进、成熟的技术路线，保证平台的稳定可靠性；同时充分考虑用户交互的便利性和界面操作的易用性，采用流行的灵活且易扩展系统架构，使用先进的技术框架路线进行设计与开发，支持基于 API 调用各组件协同工作，能够满足系统纵向集成与横向整合的信息交互。
 - 采用可靠的技术架构，支持系统稳定运行，每个组件都能提供高可用性，能够保障关键组件冗余与高可用。
 - 能够将自身运行的内在状态数据，通过采集、分析、处理后，通过合理的聚合，归纳为指标数据，方便运维人员在最短时间内了解到系统运行实时状态，同时可提供对应的 API 供其它系统读取状态指标数据。

- 生命周期管理

- 支持图形化或通过 YAML 创建、更新和删除 RabbitMQ 实例。
- 支持热部署、热更新，除通用技术服务组件升级、平台升级等重大升级改造外，系统部署和升级无需停机。

- 图形化界面操作

支持以图形化方式创建、查看、更新、删除以及弹性扩容。管理员可以根据自己的偏好，定制图形化参数。

- 容器化消息平台

RabbitMQ 基于 Kubernetes 和 Docker，原生支持容器化部署，将资源利用率提高到最大。

- 兼容性和开放性

- RabbitMQ 兼容 Intel CPU 和国产化 CPU（如飞腾 ARM，华为鲲鹏 ARM 和海光 x86 等），兼容国产操作系统（如麒麟操作系统 v10，统信服务器操作系统 UOS 等），这些均有授权的电子证明和证书。

- 具有开放的体系架构，提供开放标准 API 接口保证基础能力、管理服务、日志、监控等资源和服务被高效的调度、管理与使用，支持第三方管理平台通过 API 接口等方式实现相关编排与使用。

适用场景

RabbitMQ 适用的场景广泛，本节列举了几个典型场景。

异步处理

场景说明：用户注册后，需要发送注册邮件和注册短信。

引入消息队列后，用户的响应时间就等于写入数据库的时间 + 写入消息队列的时间（这个可以忽略不计）。引入消息队列后处理后，响应时间是串行的 3 倍，是并行的 2 倍。

异步处理

异步处理

应用解耦

场景说明：在促销活动时用户下单数量激增，用户下单后，订单系统需要通知库存系统。

传统的做法就是订单系统调用库存系统的接口，这会导致库存系统出现故障时，订单会失败。如果使用消息队列（如下图），用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功。订阅下单的消息，获取下单消息，进行库操作。就算库存系统出现故障，消息队列也能保证消息的可靠投递，不会导致消息丢失。

应用解耦

应用解耦

流量削峰

场景说明：某平台策划的秒杀活动，一般会因为流量过大，导致应用不可用。

为了解决这个问题，一般在应用前端加入消息队列。通过消息队列可以控制活动人数，超过此一定阀值的订单直接丢弃，同时可以缓解短时间的高流量压垮应用。服务器收到用户的请求之后，首先写入消息队列，假如消息队列长度超过最大值，则直接抛弃用户请求或跳转到错误页面。秒杀业务根据消息队列中的请求信息，再做后续处理。

流量削峰

流量削峰

离线升级中间件 - RabbitMQ 模块

本页说明从[下载中心](#)下载中间件 - RabbitMQ 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `_mcamel_` 字样是中间件模块的内部开发代号。

从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 `chart-syncer`

同步镜像到镜像仓库，该方法更加高效便捷。

chart-syncer 同步镜像到镜像仓库

1. 创建 `load-image.yaml`

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

==== “已安装 chart repo”

若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mccamel-offline # 到执行 charts-syncer 命令的相对路径,
 而不是此 YAML 文件和离线包之间的相对路径
 target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
 repo:
 kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
 url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
 auth:
```

```

username: "admin" # 你的镜像仓库用户名
password: "Harbor12345" # 你的镜像仓库密码
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```

```

==== “未安装 chart repo”

若当前环境未安装 chart repo, chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```

```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: LOCAL
 path: ./local-repo # chart 本地路径
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```

```

2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```

tar -xvf mcamel-rabbitmq_0.13.1_amd64.tar
cd mcamel-rabbitmq_0.13.1_amd64
tar -xvf mcamel-rabbitmq_0.13.1.bundle.tar

```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar

- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

!!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

==== “通过 helm repo 升级”

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep rabbitmq
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-rabbitmq http://{harbor url}/chartrepo/{project}
```

```

1. 更新 helm 仓库。

```
```shell
helm repo update mcamel/mcamel-rabbitmq # helm 版本过低会导致失败，若失败，请尝试
执行 helm update repo
```

```

```
```
```

- 选择您想安装的版本（建议安装最新版本）。

```
```shell
helm search repo mcamel/mcamel-rabbitmq --versions
```
```
```none
[root@master ~]# helm search repo mcamel/mcamel-rabbitmq --versions
NAME CHART VERSION APP VERSION DESCRIPTION
mcamel/mcamel-rabbitmq 0.13.1 0.13.1 A Helm chart for Kubernetes
...
```
```

```

- 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-rabbitmq -n mcamel-system -o yaml > mcamel-rabbitmq.yaml
```
```

```

- 执行 `helm upgrade`。

升级前建议您覆盖 mcamel-rabbitmq.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```
```
```shell
helm upgrade mcamel-rabbitmq mcamel/mcamel-rabbitmq \
-n mcamel-system \
-f ./mcamel-rabbitmq.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.13.1
```
```

```

== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mccamel-rabbitmq -n mccamel-system -o yaml > mccamel-rabbitmq.yaml
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```

```
```shell
helm upgrade mccamel-rabbitmq . \
-n mccamel-system \
-f ./mccamel-rabbitmq.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```

```

基本概念

本节列出有关 RabbitMQ 涉及的专有名词及术语，方便您更好地理解相关概念并使用 RabbitMQ 消息队列。

- **消息 (Message)**

消息一般分为两部分：消息体和标签。标签也称为消息头，主要用来描述这条消息。

消息体是消息的内容，是一个 json 体或者数据等。

消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括 routing-key (路由键)、priority (相对于其他消息的优先权)、delivery-mode (指出该消息可能需要持久性存储) 等。

生产者发布消息，消费者使用消息，生产者和消费者彼此并无直接关系。

- 消息标识（Message ID）

消息标识是消息的可选属性，类型为一个短字符串（short string）。

- 队列（Queue）

队列用于存储消息，生产者将消息送到队列，消费者从队列中获取消息。多个消费者

可以同时订阅同一个队列，队列里的消息分配给不同的消费者。每个消息都会被投
入到一个或多个队列里。

- 消息存活时间

消息在队列中的有效期。某条消息在队列中的留存时间超过配置的消息存活时间时，
则该消息过期。消息存活时间的值必须为非负整型数，单位为毫秒。例如，某条消
息的存活时间的值是 1000，则代表该消息最多会在队列中存活 1 秒。

- 延时消息

生产者将消息发布到消息队列 RabbitMQ 版服务端，但并不期望这条消息立马投递，
而是延迟一定时间后才投递到消费者进行消费，该消息即延时消息。

- 生产者（Publisher）

消息的生产者，也是一个向交换器发布消息的客户端应用程序。即向队列发布消息的一方。发布消息的最终目的在于将消息内容传递给其他系统/模块，使对方按照约
定处理该消息。

- 消费者（Consumer）

消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。消费者订阅
RabbitMQ 的队列。当消费者使用一条消息时，只是使用消息的消息体。在消息路
由的过程中，会丢弃标签，存入到队列中的只有消息体。

- 代理 (Broker)

消息中间件的服务节点，即消息队列服务器实体。

RabbitMQ Release Notes

本页列出 RabbitMQ 消息队列的 Release Notes，便于您了解各版本的演进路径和特性变化。

*[mcamel-rabbitmq]: mcamel 是 DaoCloud 所有中间件的开发代号，rabbitmq 实现了高级消息队列协议 (AMQP) 的消息代理软件

2024-09-30

v0.24.0

- 修复 部分操作无审计日志的问题
- 修复 修改实例集群名称后导致无法查询监控数据的问题

2024-08-31

v0.23.0

- 优化 创建实例时不可选择异常的集群

2024-05-31

v0.11.0

- 新增 参数模版

- **新增** 参数模板导入、导出功能

2024-04-30

v0.19.0

- **优化** 增加命名空间配额的提示

2024-03-31

v0.18.0

- **优化** 中文模式下支持中文的监控面板

2024-01-31

v0.17.0

- **优化** 在全局管理中增加 RabbitMQ 版本展示

2023-12-31

v0.16.0

- **修复** 创建实例时部分输入框填写特殊字符的校验未生效的问题

2023-11-30

v0.15.0

- **新增** Mcamel-RabbitMQ 支持访问白名单设置

- **新增** 支持记录操作审计日志
- **优化** 实例列表未获取到列表信息时的提示

2023-10-31

v0.14.0

- **新增** 离线升级
- **新增** 实例重启功能
- **修复** cloudshell 权限问题

2023-08-31

v0.13.0

- **优化** KindBase 语法兼容
- **优化** operator 创建过程的页面展示
- **优化** 在创建页面添加默认反亲和配置

2023-07-31

v0.12.3

- **新增** 工作负载反亲和能力
- **新增** UI 界面的权限访问限制

2023-06-30**v0.11.0**

- 新增 LoadBalancer 服务类型
- 优化 监控图表，去除干扰元素并新增时间范围选择

2023-04-27**v0.10.1**

- 新增 mcamel-rabbitmq 详情页面展示相关的事件
- 新增 mcamel-rabbitmq 支持自定义角色
- 新增 mcamel-rabbitmq 支持接入链路追踪
- 优化 mcamel-rabbitmq 调度策略增加滑动按钮

2023-03-30**v0.9.1**

- 优化 mcamel-rabbitmq 改进 Operator 镜像为加速地址

2023-03-28**v0.9.0**

- 新增 mcamel-rabbitmq 支持中间件链路追踪适配
- 新增 在安装 mcamel-rabbitmq 根据参数配置启用链路追踪

2023-02-23**v0.8.0**

- 新增 `mcamel-rabbitmq helm-docs` 模板文件
- 新增 `mcamel-rabbitmq` 应用商店中的 Operator 只能安装在 `mcamel-system`
- 新增 `mcamel-rabbitmq` 支持 cloud shell
- 新增 `mcamel-rabbitmq` 支持导航栏单独注册
- 新增 日志查看操作说明，支持自定义查询、导出等功能
- 升级 `mcamel-rabbitmq` 升级离线镜像检测脚本
- 新增 `mcamel-rabbitmq` 支持查看日志
- 修复 `mcamel-rabbitmq` 实例名太长导致自定义资源无法创建的问题
- 修复 `mcamel-rabbitmq` 工作空间 Editor 用户无法查看实例密码

2022-12-25**v0.7.0**

- 新增 `mcamel-rabbitmq NodePort` 端口冲突提前检测
- 新增 `mcamel-rabbitmq` 节点亲和性配置
- 优化 `mcamel-rabbitmq-ui` 中间件样式走查优化

2022-11-28**v0.6.4**

- 新增 获取用户列表接口

- **新增** 支持多架构的镜像，配置方式为 `depend.arm64-img.rabbitClusterImageFormat: xxxx`
- **新增** 支持 sc 扩容拦截，当 sc 不支持扩容的时候，直接拦截掉
- **新增** 返回列表或者详情时的公共字段
- **新增** 返回 alerts
- **新增** 校验 Service 注释
- **优化** 密码校验调整为 MCamel 中等密码强度
- **修复** 页面控制台可能访问到错误的端口

2022-10-27

v0.6.1

- **新增** 增加覆盖率
- **新增** 前端的 UI 注册功能
- **新增** 性能增强
- **新增** 列表页增加分页功能
- **新增** 增加修改配置的功能
- **新增** 增加返回可修改配置项的功能
- **新增** 更改创建实例的限制为集群级别，原来为 namespace 级别
- **新增** 增加监控地址的拼接功能
- **新增** 增加可以修改版本号的功能
- **新增** 修改底层 update 逻辑为 patch 逻辑
- **新增** RabbitMQ e2e 测试覆盖率 17.24% 左右

- **新增** 增加 RabbitMQ 性能压测报告
- **新增** 增加 RabbitMQ bug 抽查
- **新增** 对接 ghippo 增加 workspace 接口
- **新增** 对接 insight 通过 crd 注入 dashboard
- **新增** 将时间戳 api 字段统一调整为 int64
- **新增** 单测覆盖率提升到 53%
- **优化** 更新 release note 脚本，执行 release-process 规范
- **新增** 新增功能说明
- **新增** 创建 RabbitMQ
- **新增** RabbitMQ 数据迁移
- **新增** 实例监控
- **新增** 首次进入 RabbitMQ
- **新增** 适用场景

创建 RabbitMQ

在 RabbitMQ 消息队列中，执行以下操作：

1. 在右上角点击 **新建实例**。

点击新建实例

点击新建实例

2. 在 **创建 RabbitMQ 实例** 页面中，设置基本信息后，点击 **下一步**。

基本信息

基本信息

3. 配置规格后，点击 **下一步**。

- 版本：RabbitMQ 的版本号，当前仅支持 RabbitMQ 3.7.20。
- 副本数：支持 1、3、5、7 副本数。
- 资源配额：根据实际情况选择规则。
- 存储卷：选择 RabbitMQ 实例的存储卷和储存空间总量。

配置规格

配置规格

4. 服务设置后，点击 **下一步**。

- 访问方式：可以选择集群内访问还是 Nodeport 访问。
- 服务设置：设置连接 RabbitMQ 实例的用户名、密码。

服务设置

服务设置

5. 确认实例信息无误，点击 **确定** 完成创建。

确认

确认

6. 在实例列表页查看实例是否创建成功。刚创建的实例状态为 **未就绪**，等几分钟后该

状态变为 **运行中**。

状态

状态

更新 RabbitMQ

如果想要更新或修改 RabbitMQ 的资源配置，可以按照本页说明操作。

1. 在消息队列中，点击右侧的 **...** 按钮，在弹出菜单中选择 **更新实例**。

选择 **更新实例**

选择 **更新实例**

2. 修改基本信息后，点击 **下一步**。

基本信息

基本信息

3. 修改规格配置后，点击 **下一步**。

规格配置

规格配置

4. 修改服务设置后，点击 **确定**。

服务设置

服务设置

5. 返回消息队列，屏幕右上角将显示消息：**更新实例成功**。

成功

成功

删除 RabbitMQ

如果想要删除一个消息队列，可以执行如下操作：

1. 在消息队列中，点击右侧的 **...** 按钮，在弹出菜单中选择 **删除实例**。

选择 **删除实例**

选择 **删除实例**

2. 在弹窗中输入该消息队列的名称，确认无误后，点击 **删除** 按钮。

点击删除

点击删除

!!! warning

删除实例后，该实例相关的所有消息也会被全部删除，请谨慎操作。

查看消息队列

本节说明如何查看 RabbitMQ 消息队列。

1. 在消息队列页面中，点击某个名称。

点击某个名称

点击某个名称

2. 进入消息队列概览，查看访问设置、资源配额和 Pod 列表等信息。

查看

查看

RabbitMQ 数据迁移

RabbitMQ 的数据包括元数据（RabbitMQ 用户、vhost、队列、交换和绑定）和消息数据，

其中消息数据存储在单独的消息存储库中。

由于业务需要，要求把 `rabbitmq-cluster-a` 集群上的数据迁移到 `rabbitmq-cluster-b` 集群上。

数据迁移步骤

!!! info

从 V3.7.0 开始，RabbitMQ 将所有消息数据存储在 `_msg_stores/vhosts` 目录中，并存储在每个 vhost 的子目录中。

每个 vhost 目录都以哈希命名，并包含一个带有 vhost 名称的 `.vhost` 文件，因此可以单独备份特定 vhost 的消息集。

了解[更多信息](<https://www.rabbitmq.com/backup.html>)。

RabbitMQ 数据迁移，可以采用如下两种方案：

- 方案一：不迁移数据，先切换生产端，再切换消费端。
- 方案二：先迁移数据，然后同时切换生产端和消费端。

方案一

不迁移数据，先切换生产端，再切换消费端。

操作流程

1. 将消息生产端切换到集群 `rabbitmq-cluster-b`，不再生产消息到 `rabbitmq-cluster-a` 集群中。
2. 消费端同时消费 `rabbitmq-cluster-a` 和 `rabbitmq-cluster-b` 集群中的消息。当 `rabbitmq-cluster-a` 集群中消息全部消费完后，将消息消费端切换到 `rabbitmq-cluster-b` 集群中，完成数据迁移。

验证方法

- 在 RabbitMQ Management Web UI 页面查看。

查看

查看

- 调用 API 查看

```
curl -s -u username:password -XGET http://ip:port/api/overview
```

参数说明：

- username：使用 **rabbitmq-cluster-a** 集群的 RabbitMQ Management WebUI 的帐号
 - password：使用 **rabbitmq-cluster-a** 集群的 RabbitMQ Management WebUI 的密码
 - ip：使用 **rabbitmq-cluster-a** 集群的 RabbitMQ Management WebUI 的 IP 地址
 - port：使用 **rabbitmq-cluster-a** 集群的 RabbitMQ Management WebUI 的端口号
- 在 Overview 视图中，消费消息数（Ready）以及未确定的消息数（Unacked）都为 0，说明消费完成。

消息数为 0

消息数为 0

方案二

先迁移数据，然后同时切换生产端和消费端。借助 shovel 插件完成数据迁移。

shovel 迁移数据的原理是消费 **rabbitmq-cluster-a** 集群中的消息，将消息生产到 **rabbitmq-cluster-b** 集群中，迁移后 **rabbitmq-cluster-a** 集群中的消息被清空，建议离线迁移，业务会出现中断。

rabbitmq-cluster-a 和 **rabbitmq-cluster-b** 均需要开启并配置 shovel 插件。

启用 shovel

开启插件

1. 进入 中间件 -> RabbitMQ 的实例列表，进入 rabbitmq-cluster-a 的概览页面点击 控制台 按钮；

控制台

控制台

2. 执行以下命令，该过程可能会持续一两分钟：

```
rabbitmq-plugins enable rabbitmq_shovel rabbitmq_shovel_management
```

执行命令

执行命令

3. 进入 RabbitMQ 管理平台，在 admin 页签下可以看到 shovel 相关的插件信息。

开启插件

开启插件

进入实例 rabbitmq-cluster-b 的概览页面，再次执行以上操作。

基本配置

插件配置

插件配置

参数说明：

- Name: 配置 shovel 的名称。
- Source: 指定协议类型、连接的源集群地址，源端的类型。
- Prefetch count: 表示 shovel 内部缓存（从源端集群到目的实例之间的缓存部分）的消息数。

息条数。

- Auto-delete: 默认为 Never , 表示不删除本集群消息 , 如果设置为 After initial length transferred , 则在消息转移完成后删除。
- Destination: 指定协议类型 , 连接目标集群地址 , 目标端的类型。
- Add forwarding headers: 设置为 true , 则会在转发的消息内添加 x-shovelled 的 header 属性。
- Reconnect delay : 指定在 Shovel link 失效的情况下 , 重新建立连接前需要等待的时间 , 单位为秒。如果设置为 0 , 则不会进行重连动作 , 即 Shovel 会在首次连接失效时停止工作。默认为 5 秒。
- Acknowledgement mode : 参考 Federation 的配置。
 - no ack 表示无须任何消息确认行为 ;
 - on publish 表示 Shovel 会把每一条消息发送到目的端之后再向源端发送消息确认 ;
 - on confirm 表示 Shovel 会使用 publisher confirm 机制 , 在收到目的端的消息确认之后再向源端发送消息确认。

!!! note

服务地址 (图中的 3 和 4) 设置格式: amqp://用户名:密码@{rabbitmq 服务地址}

下图是一个简单的配置示例

配置示例

配置示例

启动迁移

迁移任务将自动启动 , 当 shovel 状态为 running 时 , 表示迁移开始 , 如下图所示。

运行状态

运行状态

迁移前后观察两个集群的队列状态，可明显看到数据迁移变化：

- 迁移启动前 **rabbitmq-cluster-a** 集群消息情况。

迁移前消息

迁移前消息

- 迁移启动后 **rabbitmq-cluster-a** 集群消息情况，可见队列消息已迁出。

集群消息

集群消息

- 迁移启动后 **rabbitmq-cluster-b** 集群消息情况，可见队列消息已迁入该集群。

集群消息

集群消息

数据迁移完成后，即可将生产端、消费端切换至 **rabbitmq-cluster-b** 集群中，完成迁移过
程。

实例监控

RabbitMQ 内置了 Prometheus 和 Grafana 监控模块。

1. 在消息队列页面中，点击某个名称。

点击某个名称

点击某个名称

2. 在左侧导航栏，点击 实例监控，可以接入监控模块。

实例监控

实例监控

各项监控指标如下。

| Panel 名 | 指标名称 | 说明 |
|--------------|------------|---------------------------------------|
| connections | 连接数 | 该指标用于统计 RabbitMQ 实例中的总连接数。 |
| channels | 通道数 | 该指标用于统计 RabbitMQ 实例中的总通道数。 |
| queues | 队列数 | 该指标用于统计 RabbitMQ 实例中的总队列数。 |
| consumers | 消费者数 | 该指标用于统计 RabbitMQ 实例中的总消费者数。 |
| publish | 生产速率 | 统计 RabbitMQ 实例中实时消息生产速率。 |
| socket_used | Socket 连接数 | 该指标用于统计当前节点 RabbitMQ 所使用的 Socket 连接数。 |
| CPU Usage | CPU 使用量 | 该指标用于统计节点 CPU 使用量。 |
| Memory Usage | 内存使用量 | 该指标用于统计节点内存使用量。 |

查看 RabbitMQ 日志

操作步骤

通过访问每个 RabbitMQ 的实例详情，页面；可以支持查看 RabbitMQ 的日志。

1. 在 RabbitMQ 实例列表中，选择想要查看的日志，点击 **实例名称** 进入到实例详情页面。



2. 在实例的左侧菜单栏，会发现有一个日志查看的菜单栏选项。

image
image

3. 点击 **日志查看** 即可进入到日志查看页面（[Insight](#) 日志查看）。

日志查看说明

在日志查看页面，我们可以很方便的进行日志查看，常用操作说明如下：

- 支持 **自定义日志时间范围**，在日志页面右上角，可以方便地切换查看日志的时间范围（可查看的日志范围以 可观测系统设置内保存的日志时长为准）
- 支持 **关键字检索日志**，左侧检索区域支持查看更多的日志信息
- 支持 **日志量分布查看**，中上区域柱状图，可以查看在时间范围内的日志数量分布
- 支持 **查看日志的上下文**，点击右侧 **上下文** 图标即可
- 支持 **导出日志**

image
image

为 RabbitMQ 添加自定义插件

问题描述

RabbitMQ 有很多插件，但是在安装 RabbitMQ 时，只能安装默认的插件，如果需要安装其他插件，需要在安装完成后，手动安装。

解决方案

在创建的 RabbitMQ 的 yaml 中，增加 initContainer，用于下载插件，然后将插件挂载到 RabbitMQ 的插件目录中，最后在 rabbitmq 的 config 中，增加了插件的配置，并在启用

的插件模块中启用插件。

示例

这里以 rabbitmq_message_timestamp-3.8.0.ez 为例，主要做的内容如下：

- 增加了 initContainer，用于下载插件
- 然后将插件挂在到 RabbitMQ 的插件目录中
- 在 rabbitmq 的 config 中，增加了插件的配置
- 并在启用的插件模块中启用插件

修改示例实例代码

```
-  
  ````yaml  
 apiVersion: rabbitmq.com/v1beta1
 kind: RabbitmqCluster
 metadata:
 ...
 status:
 ...
 spec:
 image: docker.m.daocloud.io/library/rabbitmq:3.9.25-management-alpine
 override:
 service:
 spec:
 ports:
 - name: amqp
 port: 5672
 protocol: TCP
 targetPort: 5672
 - name: management
 port: 15672
 protocol: TCP
 targetPort: 15672
 - name: prometheus
 port: 15692
 protocol: TCP
 targetPort: 15692
```

```
statefulSet:
 spec:
 template:
 spec:
 # 以下为增加部分
 + volumes:
 + - name: community-plugins
 + emptyDir: {}
 + initContainers:
 + - command:
 + - sh
 + - -c
 + - curl -L -v https://github.com/rabbitmq/rabbitmq-message-timestamp/releases/download/v3.8.0/rabbitmq_message_timestamp-3.8.0.ez --output rabbitmq_message_timestamp-3.8.0.ez
 + image: docker.m.daocloud.io/curlimages/curl:7.70.0
 + imagePullPolicy: IfNotPresent
 + name: copy-community-plugins
 + resources:
 + limits:
 + cpu: 100m
 + memory: 500Mi
 + requests:
 + cpu: 100m
 + memory: 500Mi
 + terminationMessagePolicy: FallbackToLogsOnError
 + volumeMounts:
 + - mountPath: /community-plugins/
 + name: community-plugins
 # 以上为增加部分
 containers:
 - name: rabbitmq
 ports:
 - containerPort: 5672
 name: amqp
 protocol: TCP
 - containerPort: 15672
 name: management
 protocol: TCP
 - containerPort: 15692
 name: prometheus
 protocol: TCP
 resources: {}
 # 以下为增加部分
```

```

+ volumeMounts:
+ - mountPath: /opt/rabbitmq/community-plugins
+ name: community-plugins
以上为增加部分

persistence:
 storage: 1Gi
 storageClassName: hwameistor-storage-lvm-hdd

rabbitmq:
+ envConfig: |
+ PLUGINS_DIR=/opt/rabbitmq/plugins: /opt/rabbitmq/community-plugins
以上一行行为增加部分

additionalConfig: |

 log.console.level = info
 default_user=rabbitmq
 default_pass=UE81O6Y4^w$!WP86g
 cluster_partition_handling = pause_minority
 vm_memory_high_watermark.paging_ratio = 0.99
 disk_free_limit.relative = 1.0
 collect_statistics_interval = 10000

additionalPlugins:
+ - rabbitmq_message_timestamp # 增加次插件启用
- rabbitmq_peer_discovery_k8s
- rabbitmq_prometheus
- rabbitmq_management

replicas: 1

resources:
 limits:
 cpu: 200m
 memory: 512Mi
 requests:
 cpu: 200m
 memory: 512Mi

secretBackend: {}

service:
 type: ClusterIP
 terminationGracePeriodSeconds: 604800
 tls: {}

这里的插件采用 github 官方作为下载地址，生成使用，建议自行维护插件位置，以免下
载失败。

```

## 注意事项

目前支持手工在 YAML 编辑自定义资源的方式增加对应的插件，存在一定的操作风险性，建议谨慎操作。

## 什么是 RocketMQ

RocketMQ 是一款低延迟、高并发、高可用、高可靠的分布式消息中间件。RocketMQ 可以为分布式应用系统提供异步解耦和削峰填谷的能力，同时也具备互联网应用所需的海量消息堆积、高吞吐、伸缩灵活等特性。

RocketMQ 已经成为业内共识的金融级可靠业务消息首选方案，被广泛应用于互联网、大数据、移动互联网、物联网等领域的业务场景。

参阅 [RocketMQ 与 ActiveMQ 及 Kafka 消息队列的对比](#)。

DaoCloud 在开源 Apache RocketMQ 做了容器化的定制开发，提供了简单易用的 UI 界面，可以轻松部署 RocketMQ 集群处理消息业务。

```
rocketmq ui
rocketmq ui
```

## RocketMQ 基本概念

本页列出一些 RocketMQ 的基本概念，以便您更好地理解和使用 RocketMQ。

### 主题 ( Topic )

这是 RocketMQ 中消息传输和存储的顶层容器，用于标识同一类业务逻辑的消息。主题通过 TopicName 来做唯一标识和区分。

## 消息类型 ( **MessageType** )

这是在 RocketMQ 中按照消息传输特性的不同而定义的分类，用于类型管理和安全校验。

RocketMQ 支持的消息类型有普通消息、顺序消息、事务消息和定时/延时消息。

!!! info “信息”

RocketMQ 从 5.0 版本开始，支持强制校验消息类型，即每个主题 Topic 只允许发送一种消息类型的消息，

这样可以更好的运维和管理生产系统，避免混乱。但同时保证向下兼容 4.x 版本行为，强制校验功能默认关闭，

推荐通过服务端参数 enableTopicMessageTypeCheck 手动开启校验。

## 消息队列 ( **MessageQueue** )

消息队列是 RocketMQ 中消息存储和传输的实际容器，也是消息的最小存储单元。

RocketMQ 的所有主题都是由多个队列组成，以此实现队列数量的水平拆分和队列内部的流式存储。队列通过 QueueId 来做唯一标识和区分。

## 消息 ( **Message** )

消息是 RocketMQ 中的最小数据传输单元。[生产者](#)将业务数据的负载和拓展属性包装成消息发送到服务端，服务端按照相关语义将消息投递到消费端进行消费。

## 消息视图 ( **MessageView** )

消息视图是 RocketMQ 面向开发视角提供的一种消息只读接口。通过消息视图可以读取消息内部的多个属性和负载信息，但是不能对消息本身做任何修改。

## 消息标签 ( **MessageTag** )

消息标签是 RocketMQ 提供的细粒度消息分类属性，可以在主题层级之下做消息类型的细

分。 消费者通过订阅特定的标签来实现细粒度过滤。

## 消息位点 ( **MessageQueueOffset** )

消息按到达 RocketMQ 服务端的先后顺序存储在指定主题的多个队列中，每条消息在队列中都有一个唯一的 Long 类型坐标，这个坐标被定义为消息位点。

## 消费位点 ( **ConsumerOffset** )

一条消息被某个消费者消费完成后不会立即从队列中删除，RocketMQ 会基于每个消费者分组记录消费过的最新一条消息的位点，即消费位点。

## 消息索引 ( **MessageKey** )

消息索引是 RocketMQ 提供的面向消息的索引属性。通过设置的消息索引可以快速查找到对应的消息内容。

## 生产者 ( **Producer** )

生产者是 RocketMQ 系统中用来构建并传输消息到服务端的运行实体。生产者通常被集成在业务系统中，将业务消息按照要求封装成消息并发送至服务端。

## 事务检查器 ( **TransactionChecker** )

RocketMQ 中生产者用来执行本地事务检查和异常事务恢复的监听器。事务检查器应该通过业务侧数据的状态来检查和判断事务消息的状态。

## 事务状态 ( TransactionResolution )

RocketMQ 中事务消息发送过程中，事务提交的状态标识，服务端通过事务状态控制事务消息是否应该提交和投递。 事务状态包括事务提交、事务回滚和事务未决。

## 消费者分组 ( ConsumerGroup )

消费者分组是 RocketMQ 系统中承载多个消费行为一致的消费者的负载均衡分组。 和消费者不同，消费者分组并不是运行实体，而是一个逻辑资源。在 RocketMQ 中， 通过消费者分组内初始化多个消费者实现消费性能的水平扩展以及高可用容灾。

## 消费者 ( Consumer )

消费者是 RocketMQ 中用来接收并处理消息的运行实体。消费者通常被集成在业务系统中，从服务端获取消息，并将消息转化成业务可理解的信息，供业务逻辑处理。

## 消费结果 ( ConsumeResult )

RocketMQ 中 PushConsumer 消费监听器处理消息完成后返回的处理结果， 用来标识本次消息是否正确处理。消费结果包含消费成功和消费失败。

## 订阅关系 ( Subscription )

订阅关系是 RocketMQ 系统中消费者获取消息、处理消息的规则和状态配置。 订阅关系由消费者分组动态注册到服务端系统，并在后续的消息传输中按照订阅关系定义的过滤规则进行消息匹配和消费进度维护。

## 消息过滤

消费者可以通过订阅指定消息标签（ Tag ）对消息进行过滤，确保最终只接收被过滤后的消息合集。过滤规则的计算和匹配在 RocketMQ 的服务端完成。

## 重置消费位点

以时间轴为坐标，在消息持久化存储的时间范围内，重新设置消费者分组对已订阅主题的消费进度，设置完成后消费者将接收设定时间点之后，由生产者发送到 RocketMQ 服务端的消息。

## 消息轨迹

在一条消息从生产者发出到消费者接收并处理过程中，由各个相关节点的时间、地点等数据汇聚而成的完整链路信息。通过消息轨迹，您能清晰定位消息从生产者发出，经由 RocketMQ 服务端，投递给消费者的完整链路，方便定位排查问题。

## 消息堆积

生产者已经将消息发送到 RocketMQ 的服务端，但由于消费者的消费能力有限，未能在短时间内将所有消息正确消费掉，此时在服务端保存着未被消费的消息，该状态即消息堆积。

## 事务消息

事务消息是 RocketMQ 提供的一种高级消息类型，支持在分布式场景下保障消息生产和本地事务的最终一致性。

## 定时/延时消息

定时/延时消息是 RocketMQ 提供的一种高级消息类型，消息被发送至服务端后，在指定时间后才能被消费者消费。通过设置一定的定时时间可以实现分布式场景的延时调度触发效果。

## 顺序消息

顺序消息是 RocketMQ 提供的一种高级消息类型，支持消费者按照发送消息的先后顺序获取消息，从而实现业务场景中的顺序处理。

更多请参阅 [Apache RocketMQ 官方文档](#)。

# 离线升级中间件 - RocketMQ 模块

本页说明从[下载中心](#)下载中间件 - RocketMQ 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `_mcamel_` 字样是中间件模块的内部开发代号。

## 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 `chart-syncer` 同步镜像到镜像仓库，该方法更加高效便捷。

### chart-syncer 同步镜像到镜像仓库

1. 创建 `load-image.yaml`

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

### ==== “已安装 chart repo”

若当前环境已安装 chart repo, chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
  url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```
```

```

==== “未安装 chart repo”

若当前环境未安装 chart repo, chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```
```yaml title="load-image.yaml"
source:
 intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
 containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
 containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
 kind: LOCAL
 path: ./local-repo # chart 本地路径
containers:
 auth:
 username: "admin" # 你的镜像仓库用户名
 password: "Harbor12345" # 你的镜像仓库密码
```
```

```

## 2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

1. 解压 tar 压缩包。

```
tar -xvf mcamel-rocketmq_0.1.0_amd64.tar
cd mcamel-rocketmq_0.1.0_amd64
tar -xvf mcamel-rocketmq_0.1.0.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

### !!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

==== “通过 helm repo 升级”

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep rocketmq
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-rocketmq http://{harbor url}/chartrepo/{project}
```

```

1. 更新 helm 仓库。

```
```shell
helm repo update mcamel/mcamel-rocketmq # helm 版本过低会导致失败，若失败，请尝试执行 helm update repo
```

```

1. 选择您想安装的版本（建议安装最新版本）。

```
```shell
helm search repo mcamel/mcamel-rocketmq --versions
```

```none
[root@master ~]# helm search repo mcamel/mcamel-rocketmq --versions
NAME          CHART VERSION   APP VERSION   DESCRIPTION
mcamel/mcamel-rocketmq    0.1.0        0.1.0        A Helm chart for Kubernetes
...
```

```

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-rocketmq -n mcamel-system -o yaml > mcamel-rocketmq.yaml
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 `mcamel-rocketmq.yaml` 中的 `global.imageRegistry` 字段为当前使用的

镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-rocketmq mcamel/mcamel-rocketmq \
-n mcamel-system \
-f ./mcamel-rocketmq.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.1.0
```

==== “通过 chart 包升级”
```

#### 1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-rocketmq -n mcamel-system -o yaml > mcamel-rocketmq.yaml
```

1. 执行 `helm upgrade` 。
```

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-rocketmq . \
-n mcamel-system \
-f ./mcamel-rocketmq.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
````
```

# RocketMQ 消息队列 Release Notes

本页列出 RocketMQ 消息队列的 Release Notes，便于您了解各版本的演进路径和特性变化。

## 2025-02-28

### v0.14.0

- **新增** 支持为 RocketMQ 实例配置静态 IP
- **优化** 支持配置 时区的环境变量
- **优化** 升级 operator 镜像
- **修复** 名称服务器重启后控制台连接错误的问题

## 2024-09-30

### v0.11.0

- **修复** 容器组列表未展示实例中所有相关的容器组
- **修复** 选择工作空间查询 Redis 列表时权限泄漏的问题
- **修复** 部分操作无审计日志的问题

## 2024-08-31

### v0.10.0

- **优化** 创建实例时不可选择异常的集群

**2024-04-30****v0.6.0**

- 优化 Broker 节点启动的 JVM 参数
- 优化 增加命名空间配额的提示

**2024-03-31****v0.5.0**

- 优化 中文模式下支持中文的监控面板

**2024-01-31****v0.4.0**

- 优化 在全局管理中增加 RocketMQ 版本展示

**2023-12-31****v0.3.0**

- 修复 exporter 与 RocketMQ 版本的兼容问题
- 修复 创建实例时部分输入框填写特殊字符的校验未生效的问题

**2023-11-30****v0.2.0**

- **新增** 支持记录操作审计日志
- **优化** 实例列表未获取到列表信息时的提示

**2023-10-31****v0.1.1**

- **新增** 支持 RocketMQ 中间件
- **新增** 离线升级

## 创建 RocketMQ 实例

本页说明创建 RocketMQ 实例的操作步骤。

1. 点击左侧导航栏选择 **中间件** -> **RocketMQ 消息队列**。

RocketMQ  
RocketMQ

2. 进入目标工作空间，点击 **新建实例**。

- 实例名称：2 ~ 40 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母开头，字母或数字结尾。
- 集群 / 命名空间：选择实例部署的位置。
- 安装环境检测：如未通过安装环境检测，需要根据提示安装组件之后方可进行下一步。

RocketMQ

## RocketMQ

3. 参考下方信息填写配置规格，然后点击 **下一步**。

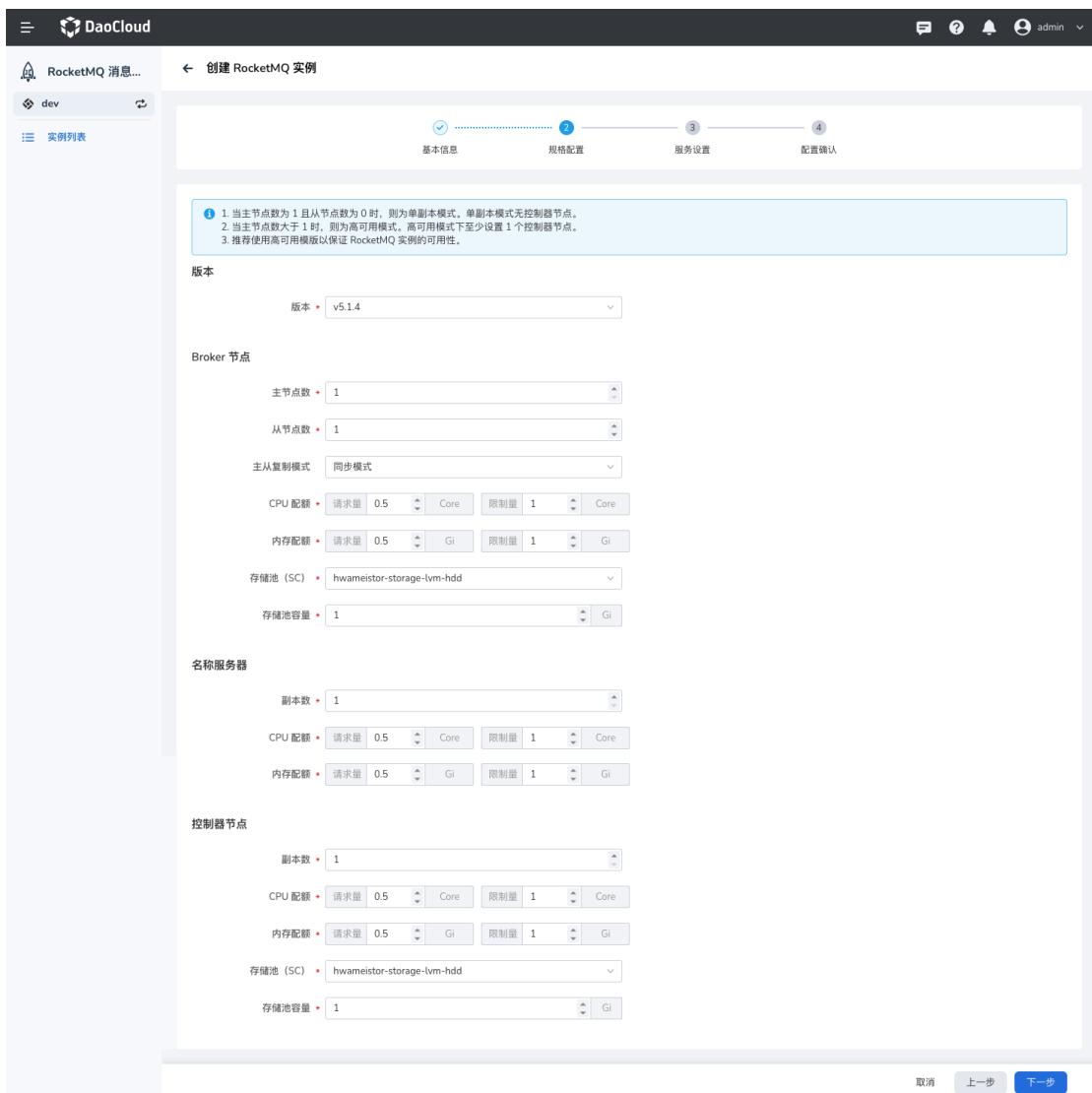
- 部署模式在实例创建之后不可更改
- 生产模式下建议采用高可用部署模式
- 高可用模式下需要至少 4 个副本
- 存储类：所选的存储类应有足够的可用资源，否则会因资源不足导致实例创建失败
- 存储容量：每个磁盘具有多少容量。实例创建之后不可调低
- 每副本磁盘数：为每个副本提供多少个次盘。实例创建之后不可调低。

### !!! note

1. 当主节点数为 1 且从节点数为 0 时，则为单副本模式。单副本模式无控制器节点。
2. 当主节点数大于 1 时，则为高可用模式。高可用模式下至少设置 1 个控制器节点。
  -
3. 推荐使用高可用模版以保证 RocketMQ 实例的可用性。

### !!! warning

当控制器节点的副本数为 3 时，会导致 Broker 节点无法启动，请勿设置控制器节点的副本数为 3！



## RocketMQ

4. 参考下方信息填写服务设置，点击 **下一步**。

- 集群内访问：只能在同一集群内部访问服务
- 节点端口：通过节点的 IP 和静态端口访问服务，支持从集群外部访问服务
- 负载均衡器：使用云服务提供商的负载均衡器使得服务可以公开访问
- 负载均衡器/外部流量策略：规定服务将外部流量路由到节点本地还是集群范围内的断点

- Cluster：流量可以转发到集群中其他节点上的 Pod

- Local：流量只能转发到本节点上的 Pod

- 控制台账号：访问此新建实例时需要用到的用户名、密码

RocketMQ

RocketMQ

5. 确认实例配置信息无误，点击 **确定** 完成创建。

RocketMQ

RocketMQ

6. 返回实例列表页查看实例是否创建成功。

创建中的实例状态为 **未就绪**，等所有相关容器成功启动之后状态变为 **运行中**。

RocketMQ

RocketMQ

## 更新 RocketMQ 实例

本页说明更新 RocketMQ 实例的操作步骤。

1. 在实例列表中，点击右侧的 ...，在弹出菜单中选择 **更新实例**。

update

update

2. 修改基本信息，然后点击 **下一步**。

- 仅支持修改描述信息

- 实例名称、部署位置不可修改

update

update

3. 修改规格配置，然后点击 **下一步**。

update

update

4. 修改服务设置，然后点击 **确定**。

update

update

5. 返回实例列表，屏幕右上角将显示消息：更新实例成功。

update

update

# 单集群跨机房高可用部署

## 场景需求

客户机房环境为单一 k8s 集群横跨 **机房 A**、**机房 B**，期望可以部署一套 3 主 3 从 RocketMQ 实现跨机房服务高可用，当任一机房整体离线时，RocketMQ 仍可以正常提供服务。

mutizone

mutizone

## RocketMQ 5.1.4 各组件镜像地址

```
docker pull ghcr.io/ksmartdata/rocketmq-controller: v5.1.4
docker pull ghcr.io/ksmartdata/rocketmq-nameserver: v5.1.4
docker pull ghcr.io/ksmartdata/rocketmq-broker: v5.1.4
```

## 解决方案

RocketMQ 5.0 通过 Dledger Controller 实现 broker 高可用自动主从切换能力，该模式通过独立的 Controller 实现主从切换，broker 节点无需遵循 craft 协议，不足半数 master 也可以进行选举。

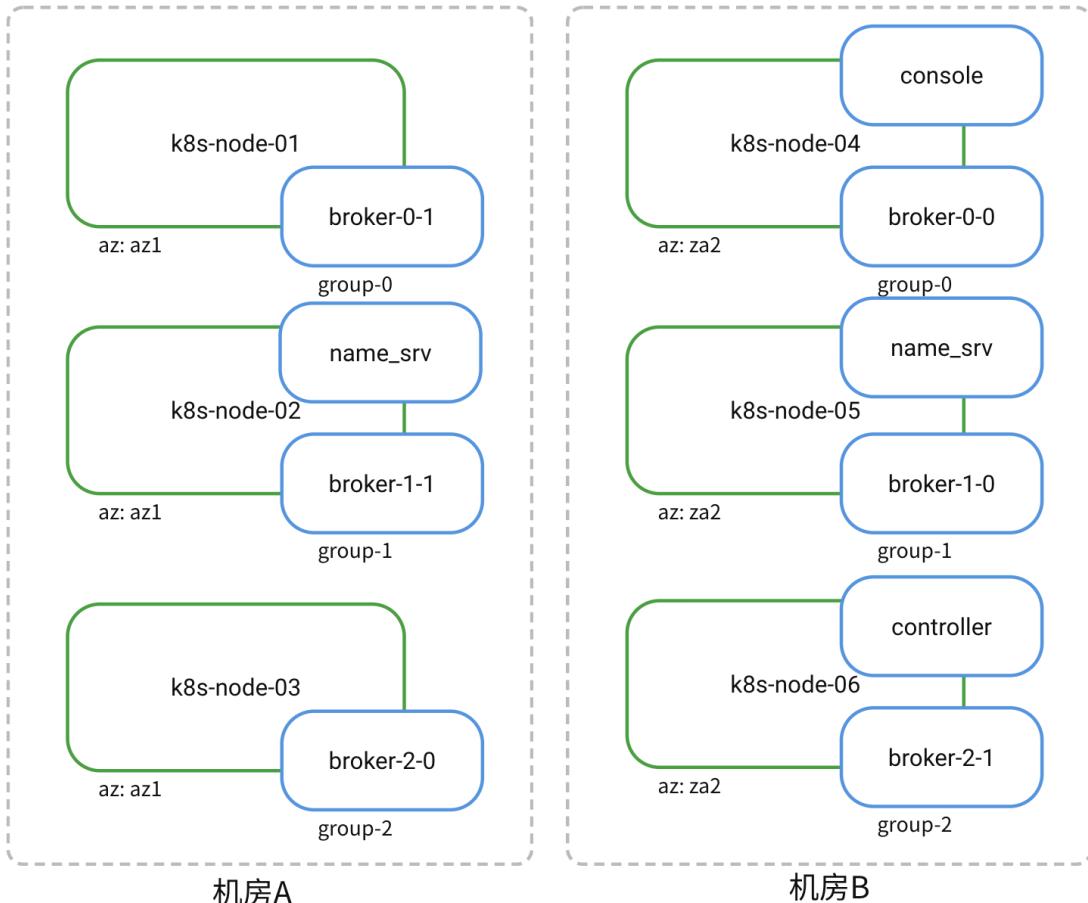
!!! warning “DCE 数据服务目前不支持 RocketMQ 5.0，请手动操作”

1. 创建完 rocektmq 集群后，需要手动将 broker cr 中的 `spec.clusterMode` 从 Static 修改为 Controller（修改完成后，Operator 应该会新建出 Controller 模式下的 Broker Pod）；
2. 删除掉 STATIC 模式下的生成的 Broker Pod 以及 PVC（Pod 名字中带有 master 和 replica 字样）。

新建的 Controller 模式的 Broker Pod 名字中没有 master 和 replica 字样。

结合 Dledger Controller 能力，做以下部署设计：

- 确保 6 个 Broker 分布在不同的集群 Node；
- 确保每一对主/从 Broker 分别分布在不同的机房；
- 尽量形成两机房的 Broker Master 1: 2 的关系，避免所有 Master 运行在同一机房
- name\_srv 在两个机房分别存在副本



mutizone

本方案采用了工作负载的调度策略，通过具有权重的节点亲和性策略和工作负载反亲和策略达成以上部署目标。

!!! info

请保证各节点资源充足，避免因资源不足导致调度器无法完成正确调度。

## 操作步骤

### 创建配置

1. 创建时键入以下配置参数：

```
inSyncReplicas=2
totalReplicas=2
minInSyncReplicas=1
enableAutoInSyncReplicas=true
mutizone
mutizone
```

2. 完成创建后，修改 CR，并重启 broker sts：

```
apiVersion: rocketmq.apache.org/v1alpha1
kind: Controller
metadata:
 name: controller
 namespace: mcamel-name-work6
spec:
 controllerImage: ghcr.io/ksmartdata/rocketmq-controller:v5.1.4
 hostPath: /data/rocketmq/controller
 imagePullPolicy: IfNotPresent
 resources:
 limits:
 memory: 2Gi
 requests:
 memory: 2Gi
 size: 1
 storageMode: StorageClass # 注意修改此项
 volumeClaimTemplates:
 - metadata:
 name: controller-storage
 spec:
 accessModes:
 - ReadWriteOnce
 resources:
 requests:
 storage: 1Gi
```

## 标签配置

### RocketMQ 工作负载标签

Broker 标签用于工作负载反亲和（自带标签，不用配置）：

| RocketMQ 组 | 工作负载                  | 标签                       |
|------------|-----------------------|--------------------------|
| group-0    | broker-0-0/broker-0-1 | broker_cr: {RocketMQ 名称} |
| group-1    | broker-1-0/broker-1-1 | broker_cr: {RocketMQ 名称} |
| group-2    | broker-2-0/broker-2-1 | broker_cr: {RocketMQ 名称} |

### 集群节点标签

为了把 broker 按预期分配在两个机房，为两个机房的节点配置不同标签，用于确保主从 broker 落在不同的机房。

| 拓扑域        | 集群节点        | 标签      |
|------------|-------------|---------|
| az<br>机房 A | k8s-node-01 | az: az1 |
|            | k8s-node-02 | az: az1 |
|            | k8s-node-03 | az: az1 |
|            | k8s-node-04 | az: az2 |
| 机房 B       | k8s-node-05 | az: az2 |
|            | k8s-node-06 | az: az2 |

## 调度配置

- RocketMQ 采用 1 个 CR 创建 6 个 broker sts ( 3 主 3 从 ) 的方式，因此每一个 broker 实例需单独配置亲和性策略。
- 每两个 broker ( broker-x-0 和 broker-x-1 ) 为一组，分别担任 master 和 slave，但并无标签可以标识角色，因此配置目标是每一组内的两 broker 运行于不同机房，以此保证主、从运行在不同的机房。

- 经观察，operator 创建 broker 的过程是按编号顺序创建，尽量避免所有 master 调度在同一机房。

配置内容如下：

### 1. 工作负载：broker-0-1 / broker-1-1 / broker-2-0

```
执行工作负载反亲和，确保每个集群节点仅调度一个 broker 副本。
affinity:
 podAntiAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 - labelSelector:
 matchExpressions:
 - key: broker_cr
 operator: In
 values:
 - {RocketMQ 名称}
 topologyKey: kubernetes.io/hostname
优先在机房 A 部署，通常每一组内先部署注册的 sts 会作为 master
nodeAffinity:
 preferredDuringSchedulingIgnoredDuringExecution:
 - weight: 100
 preference:
 matchExpressions:
 - key: az
 operator: In
 values:
 - az1
 - weight: 90
 preference:
 matchExpressions:
 - key: az
 operator: In
 values:
 - az2
```

### 2. 工作负载：broker-0-0 / broker-1-0 / broker-2-1

```
执行工作负载反亲和，确保每个集群节点仅调度一个 broker 副本。
affinity:
 podAntiAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 - labelSelector:
 matchExpressions:
```

```

 - key: broker_cr
 operator: In
 values:
 - {RocketMQ 名称}
 topologyKey: kubernetes.io/hostname
 # 与同组的 sts 相反的调度优先顺序，避免调度在同一机房
 nodeAffinity:
 preferredDuringSchedulingIgnoredDuringExecution:
 - weight: 100
 preference:
 matchExpressions:
 - key: az
 operator: In
 values:
 - az2
 - weight: 90
 preference:
 matchExpressions:
 - key: az
 operator: In
 values:
 - az1

```

### 3. 工作负载：namesrv

```

 # 两副本，每个机房部署一个 namesrv，无在线 namesrv 将导致 broker 工作异常。
 affinity:
 podAntiAffinity:
 requiredDuringSchedulingIgnoredDuringExecution:
 - labelSelector:
 matchExpressions:
 - key: name_service_cr
 operator: In
 values:
 - {namesrv 名称}
 topologyKey: az

```

### 4. 工作负载：Controller（可选）

```

 # 单副本，如果对外主要提供业务的是机房 A，可以优先部署在机房 B，该配置由具体
 # 业务情况决定。
 nodeAffinity:
 preferredDuringSchedulingIgnoredDuringExecution:
 - weight: 100
 preference:
 matchExpressions:

```

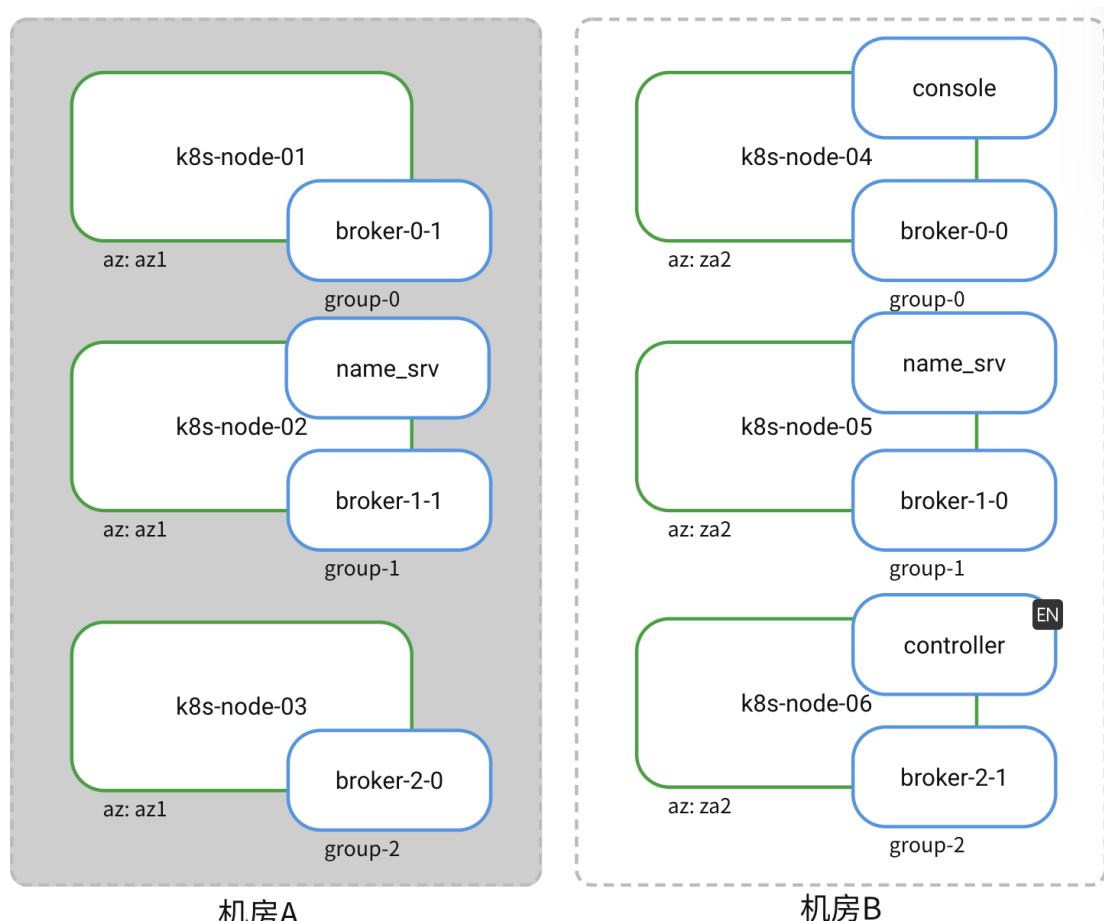
```
- key: az
 operator: In
 values:
 - az2
- weight: 90
 preference:
 matchExpressions:
 - key: az
 operator: In
 values:
 - az1
```

## 机房离线处理

经由上述部署方式，3 个主从组会把 master/slave broker 分别部署在不同机房。

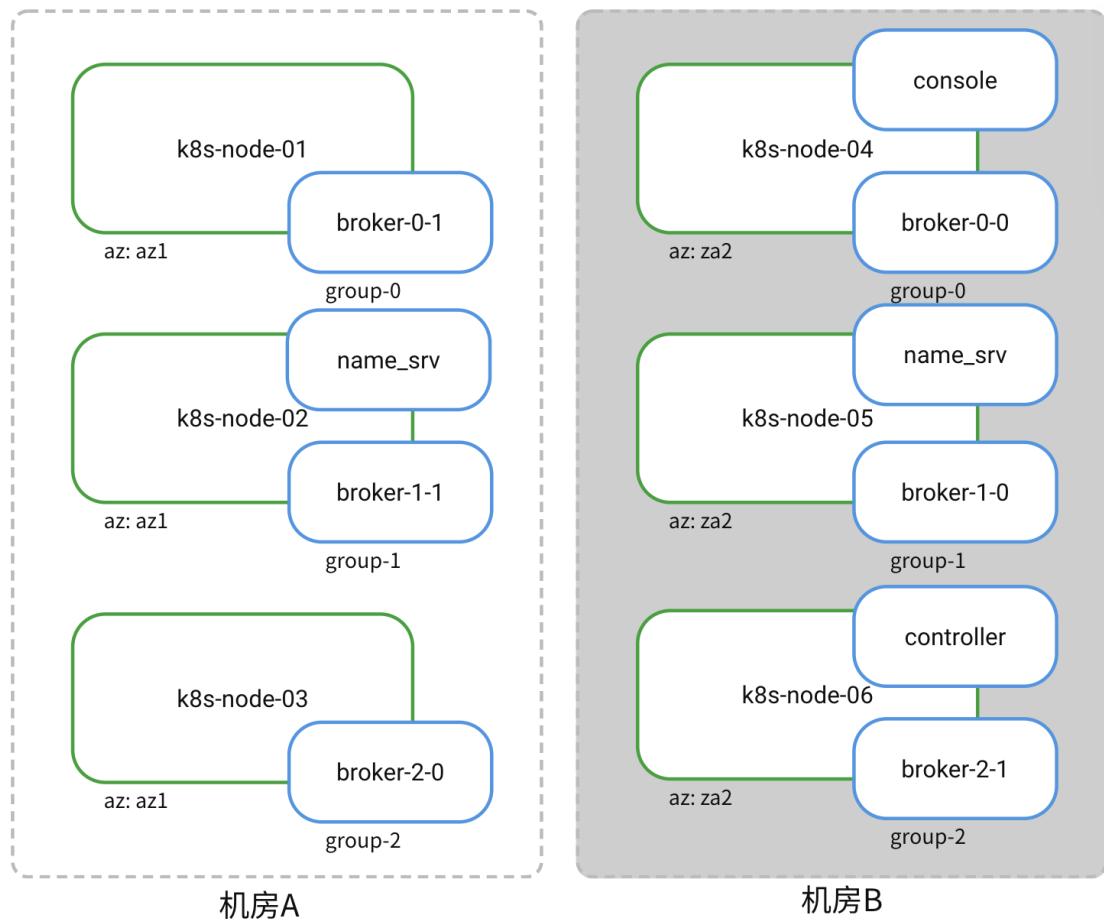
### 机房 A

Controller 运行于机房 B，基于 Dledger Controller 的主从转换机制，可以实现机房 B 的 broker slave -> master 自动升级，无需人工干预。



mutizone

## 机房 B 离线



mutizone

- 负责调度的 Controller 离线，将暂时无法 slave -> master 转换，需要手动删除 Pod 漂移至其他节点。
- Controller 重新调度至现存节点，才能继续完成机房 A 的 slave -> master 自动升级。

## 一些注意事项

1. broker 角色升级失败：经实际测试，Controller 稳定性不是很好，broker 的 slave -> master 自动升级有一定几率失败，可通过重启 Controller 的方式，即可解决该问题。

2. 谨慎使用 sts 的删除操作：删除重建 broker 会导致配置在实例中的调度策略丢失，

但不会丢失配置在 CR 的策略，因此建议谨慎使用删除操作。重启 sts 的操作不会造成以上的丢失情况。

mutizone

mutizone

3. 机房离线导致 console 无法获取数据：如果采用 2 副本 console，一个机房离线可能导致 console 无法连接 name\_svr。

解决办法：重启现存 console 工作负载。

4. dashboard 异常查看 broker 列表：如果 dashboard 不可访问，也可以通过其他方式获取 broker 列表信息，建议在部署完成后立即使用一下方式查看节点分布，确保符合预期。

- 方法 1：进入 name\_svr Pod，执行以下命令：

```
./mqadmin clusterList -n 127.0.0.1:9876
mutizone
mutizone
```

- BID = 0：表示该节点为 Master
- BID <> 0：表示该节点为 slave

- 方法 2：进入 Controller，执行以下命令：

```
./mqadmin getSyncStateSet -a 127.0.0.1:9878 -c {实例名} -b {rocketmq 主从组名}
例：./mqadmin getSyncStateSet -a 127.0.0.1:9878 -c rmq-ha-i -b rmq-ha-i-0
```

## 什么是 MinIO

[MinIO](#) 是一款热门、轻量、开源的对象存储方案，完美兼容 AWS S3 协议，友好支持 K8s。

MinIO 是专为 AI 等云原生工作负载而设计的，可以为高性能的云原生数据机器学习、大数据分析、海量存储的基础架构等提供数据工作负载。

DCE 的 MinIO 模块是基于开源 MinIO 构建的对象存储中间件，即插即用。 DaoCloud 为上海道客网络科技有限公司

其开发了简单易用的图形化界面，向用户提供计算、存储和带宽资源独占的 MinIO 专享实例。

MinIO 之所以广受欢迎，是因为其具有以下特点：

#### 1. 高性能。

MinIO 是全球领先的对象存储服务先锋，目前在全世界有数百万的用户。在标准硬件上，读/写速度高达每秒几百 GB。

#### 2. 可扩展性。

MinIO 借鉴了 Web 缩放器，为对象存储带来了简单的扩缩模型。在部署 MinIO 时，扩展从单个群集开始。

#### 3. 云原生支持。

MinIO 是在过去几年内从 0 开始打造的一款存储方案，符合一切云原生计算的架构和构建流程，并且包含最新的云计算技术和理念。

#### 4. 纯开源。

MinIO 基于 Apache V2 license 100% 开放源代码。这就意味着 MinIO 的客户能够自动、无限制、自由免费地使用和集成 MinIO、自由创新和创造、自由修改和完善、自由再次发行新版本和组合软件。

#### 5. 兼容 S3 存储。

AWS 的 S3 API（接口协议）是在全球范围内达到共识的对象存储协议，是全世界内大家都认可的标准。

#### 6. 极简。

极简主义是 MinIO 的指导性设计原则。简单性减少了出错的机会，提高了正常运行时间，提供了可靠性，同时简单性又是性能的基础。

## 7. 支持多云。

可以创建数以百万计的实例部署在私有云、公有云和边缘计算环境。

MinIO 为云原生而设计，可以作为轻量级容器运行，由外部编排服务（如 Kubernetes）管理。整个服务器约为几十 MB 的静态二进制文件，即使在高负载下也可以高效利用 CPU 和内存资源，使得企业可以在共享硬件上共同托管大量租户。

### MinIO 主界面

MinIO 主界面

[创建 MinIO 实例](#)

# 使用场景

MinIO 是一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容 AWS S3 云存储服务接口，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从 KB 到最大 TB 不等。

常见的使用场景有：

- 网盘：海量文件
- 社交网站：海量图片
- 电商网站：海量商品图片
- 视频网站：海量视频文件

每个 MinIO 集群都是分布式 MinIO 服务器的集合，每个节点一个进程。MinIO 作为一个进程在用户空间中运行，并使用轻量级的协程来实现高并发。将驱动器分组到擦除集（默认情况下，每组 16 个驱动器），然后使用确定性哈希算法将对象放置在这些擦除集上。

MinIO 专为大规模、多数据中心云存储服务而设计。每个租户都运行自己的 MinIO 集群，该集群与其他租户完全隔离，从而使租户能够免受升级、更新和安全事件的任何干扰。每个租户通过联合跨地理区域的集群来独立扩展。

## 离线升级中间件 - MinIO 模块

本页说明从[下载中心](#)下载中间件 - MinIO 模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 `_mcamel_` 字样是中间件模块的内部开发代号。

### 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 `chart-syncer` 同步镜像到镜像仓库，该方法更加高效便捷。

#### chart-syncer 同步镜像到镜像仓库

##### 1. 创建 `load-image.yaml`

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

==== “已安装 chart repo”

若当前环境已安装 `chart repo`, `chart-syncer` 也支持将 `chart` 导出为 `tgz` 文件。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
```

```

url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
auth:
  username: "admin" # 你的镜像仓库用户名
  password: "Harbor12345" # 你的镜像仓库密码
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

### ==== “未安装 chart repo”

若当前环境未安装 chart repo, chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: mcamel-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/mcamel # 需更改为你的镜像仓库
repo:
  kind: LOCAL
  path: ./local-repo # chart 本地路径
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

## 2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

### 1. 解压 tar 压缩包。

```

tar -xvf mcamel-minio_0.8.1_amd64.tar
cd mcamel-minio_0.8.1_amd64
tar -xvf mcamel-minio_0.8.1.bundle.tar

```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

#### !!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作。

加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

### ==== “通过 helm repo 升级”

1. 检查 helm 仓库是否存在。

```
```shell
helm repo list | grep minio
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加 helm 仓库。

```
```shell
helm repo add mcamel-minio http://{harbor url}/chartrepo/{project}
```

```

1. 更新 helm 仓库。

```
```shell
```

```

helm repo update mcamel/mcamel-minio # helm 版本过低会导致失败，若失败，请尝试执行 helm update repo

```

- 选择您想安装的版本（建议安装最新版本）。

```shell

```
helm search repo mcamel/mcamel-minio --versions
```
```

```

```none

| NAME | CHART VERSION | APP VERSION | DESCRIPTION |
|---------------------|---------------|-------------|-----------------------------|
| mcamel/mcamel-minio | 0.8.1 | 0.8.1 | A Helm chart for Kubernetes |
| ... | | | |
| ``` | | | |

- 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```shell

```
helm get values mcamel-minio -n mcamel-system -o yaml > mcamel-minio.yaml
```
```

```

- 执行 `helm upgrade`。

升级前建议您覆盖 mcamel-minio.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```shell

```
export imageRegistry={你的镜像仓库}
```
```

```

```shell

```
helm upgrade mcamel-minio mcamel/mcamel-minio \
-n mcamel-system \
-f ./mcamel-minio.yaml \
--set global.imageRegistry=$imageRegistry \
--version 0.8.1
```
```

```

==== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values mcamel-minio -n mcamel-system -o yaml > mcamel-minio.yaml
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```

```shell
helm upgrade mcamel-minio . \
-n mcamel-system \
-f ./mcamel-minio.yaml \
--set global.imageRegistry=${imageRegistry} \
--set console_image.registry=${imageRegistry} \
--set operator_image.registry=${imageRegistry}
```

```

## MinIO 常见术语

本页说明 MinIO 相关概念。

| 概念         | 含义                                    |
|------------|---------------------------------------|
| Object 对象  | 存储的基本对象，例如文件、图片等                      |
| Bucket 桶   | 用于存储 Object 的逻辑空间，相互之间隔离，类似于系统中的顶层文件夹 |
| Drive 驱动磁盘 | 即存储数据的磁盘，所有的对象数据都存储在 Drive 中          |

| 概念      | 含义                                                                                                                                                                                          |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Set 磁盘集 | 即一组 Drive 的集合，分布式部署根据集群规模自动划分一个或多个 Set                                                                                                                                                      |
|         | <ul style="list-style-type: none"> <li>• 一个 Object 存储在一个 Set</li> <li>• 一个集群划分为多个 Set</li> <li>• 一个 Set 中的 Drive 尽可能分布在不同的节点上</li> <li>• 一个 Set 包含的 Drive 数量是固定的，默认由系统根据集群规模自动计算</li> </ul> |
|         | 更多概念的阐述如下所述。                                                                                                                                                                                |

## 擦除码 Erasure Code

MinIO 使用按对象的嵌入式擦除编码保护数据，该编码以汇编代码编写，可提供最高的性能。 MinIO 使用 Reed-Solomon 代码将对象划分为  $n/2$  个数据和  $n/2$  个奇偶校验块，也可以将对象配置为任何所需的冗余级别。 这意味着在 12 个驱动器设置中，将一个对象碎片为 6 个数据和 6 个奇偶校验块。 即使丢失了多达  $5$  个 ( $(n / 2) - 1$ ) 个驱动器（无论是奇偶校验还是数据），仍然可以从其余驱动器可靠地重建数据。 MinIO 的实现可确保即使丢失或无法使用多个设备，也可以读取对象或写入新的对象。 最后，MinIO 的擦除代码位于对象级别，并且可以一次修复一个对象。

擦除码

擦除码

## Bitrot 保护

无声的数据损坏或 Bitrot 是磁盘驱动器面临的严重问题，导致数据在用户不知情的情况下损坏。 原因多种多样：驱动器老化，电流尖峰，磁盘固件错误，虚假写入，读/写方向错

误，驱动程序错误，意外覆盖等。但结果是一样的：数据泄漏。

MinIO 对高速哈希算法的优化实现可确保它永远不会读取损坏的数据，可以实时捕获和修复损坏的对象。通过在 READ 上计算哈希值，并在 WRITE 上从应用程序、整个网络以及内存/驱动器的哈希值来确保端到端的完整性。该实现旨在提高速度，并且可以在 Intel CPU 的单个内核上实现超过每秒 10 GB 的哈希速度。

#### Bitrot 保护

#### Bitrot 保护

## 加密

加密活跃数据与保护静态数据不同。MinIO 支持多种复杂的服务器端加密方案，以保护所有位置的存储数据。MinIO 的方法可确保机密性、完整性和真实性，而性能开销却可以忽略不计。使用 AES-256-GCM、ChaCha20-Poly1305 和 AES-CBC 支持服务器端和客户端加密。加密的对象使用 AEAD 服务器端加密进行了防篡改。此外，MinIO 与所有常用的密钥管理解决方案（例如 HashiCorp Vault）兼容并经过测试。

MinIO 使用密钥管理系统（KMS）支持 SSE-S3。如果客户端请求 SSE-S3 或启用了自动加密，则 MinIO 服务器会使用唯一的对象密钥对每个对象进行加密，该对象密钥受 KMS 管理的主密钥保护。由于开销极低，因此可以为每个应用程序和实例打开自动加密。

启用 WORM 后，MinIO 会禁用所有可能会使对象数据和元数据发生变异的 API。这意味着一旦写入数据就可以防止篡改。这对于许多不同的法规要求具有实际应用。

#### 加密

#### 加密

## 身份认证和管理

MinIO 支持身份管理中最先进的标准，并与 OpenID connect 兼容提供商以及主要的外部 IDP 供应商集成。这意味着访问是集中的，密码是临时的和轮换的，而不是存储在配置文件和数据库中。此外，访问策略是细粒度的且高度可配置的，这意味着支持多租户和多实例部署变得简单。

### 身份认证和管理

#### 身份认证和管理

## 连续复制

传统复制方法的挑战在于它们无法有效扩展到几百 TB。话虽如此，每个人都需要一种复制策略来支持灾难恢复，并且该策略需要跨越地域、数据中心和云。MinIO 的连续复制旨在用于大规模的跨数据中心部署。通过利用 Lambda 计算通知和对象元数据，它可以高效、快速地计算增量。

Lambda 通知确保与传统的批处理模式相反，更改可以立即传播。连续复制意味着即使发生高动态数据集，如果发生故障，数据丢失也将保持在最低水平。最后，就像 MinIO 所做的一样，连续复制是多厂商的，这意味着您的备份位置可以是从 NAS 到公共云的任何位置。

### 连续复制

#### 连续复制

## 全局一致性

现代企业到处都有数据。MinIO 允许将这些各种实例组合在一起以形成统一的全局命名空

间。 具体来说，最多可以将 32 个 MinIO 服务器组合成一个分布式模式集，并且可以将多个分布式模式集组合成一个 MinIO 服务器联合。 每个 MinIO Server Federation 都提供统一的管理员和命名空间。

MinIO Federation Server 支持无限数量的分布式模式集。

这种方法的影响在于，对象存储可以为大型的、地理上分散的企业进行大规模扩展，同时保留从以下位置容纳各种应用程序（S3 Select、MinSQL、Spark、Hive、Presto、TensorFlow、H2O）的能力。具有单一控制台。

#### 全局一致性

全局一致性

## 多云网关

所有企业都在采用多云策略，这也包括私有云。因此，您的裸机虚拟化容器和公共云服务（包括 Google、Microsoft 和阿里等非 S3 提供商）必须看起来完全相同。尽管现代应用程序具有高度的可移植性，但为这些应用程序提供支持的数据却并非如此。

MinIO 应对的主要挑战是：无论数据位于何处都都能使数据可用。MinIO 在裸机、网络连接存储和每个公共云上运行。更重要的是，MinIO 通过 Amazon S3 API 从应用程序和管理角度确保您对数据的看法完全相同。

MinIO 可以走得更远，使您现有的存储基础架构与 Amazon S3 兼容。其影响是深远的。

现在，组织可以从文件到块真正统一其数据基础架构，所有这些都显示为可通过 Amazon S3 API 访问的对象，而无需迁移。

#### 多云网关

## 多云网关

# MinIO 对象存储 Release Notes

本页列出 MinIO 对象存储的 Release Notes，便于您了解各版本的演进路径和特性变化。

\*[mcamel-minio]: mcamel 是 DaoCloud 所有中间件的开发代号，minio 是一款热门、轻量的对象存储中间件

**2024-09-30**

**v0.19.0**

- **修复** 选择工作空间查询 MinIO 列表时权限泄漏的问题
- **修复** 部分操作无审计日志的问题

**2024-08-31**

**v0.18.1**

- **优化** 创建实例时不可选择异常的集群
- **修复** minio operator 多副本退出机制存在缺陷，调整为单副本

**2024-07-31**

**v0.17.0**

- **修复** 集群重启后 MinIO 镜像地址错误

**2024-04-30****v0.14.0**

- 优化 增加命名空间配额的提示

**2024-03-31****v0.13.0**

- 优化 当用户权限不足时无法读取 MinIO 的密码

**2024-01-31****v0.12.0**

- 优化 MinIO 实例支持中文 Dashboard
- 优化 在全局管理中增加 MinIO 版本展示

**2023-12-31****v0.11.0**

- 新增 MinIO Operator 版本升级至 v5.0.11
- 修复 创建实例时部分输入框填写特殊字符的校验未生效的问题

**2023-11-30****v0.10.0**

- **新增** 支持记录操作审计日志
- **优化** 实例列表未获取到列表信息时的提示

**2023-10-31****v0.9.0**

- **新增** 离线升级
- **新增** 实例重启功能
- **新增** PVC 扩容
- **新增** 纳管外部实例
- **修复** Pod 列表展示地址为 Host IP
- **修复** cloudshell 权限问题

**2023-09-30****v0.8.1**

- **优化** 新增 reloc8s 文件
- **升级** Operator 镜像到 v5.0.6

**2023-08-31****v0.8.0**

- 优化 KindBase 语法兼容
- 优化 operator 创建过程的页面展示
- 升级 Operator 镜像到 v5.0.6

**2023-07-31****v0.7.3**

- 新增 UI 界面的权限访问限制

**2023-06-30****v0.7.0**

- 新增 对接全局管理审计日志模块
- 优化 监控图表，去除干扰元素并新增时间范围选择

**2023-04-27****v0.5.1**

- 新增 mcamel-minio 详情页面展示相关的事件
- 新增 mcamel-minio 支持自定义角色
- 优化 mcamel-minio 调度策略增加滑动按钮

**2023-03-28****v0.4.1**

- **修复** `mcamel-minio` 页面展示 LoadBalancer 地址错误
- **修复** `mcamel-minio` 删除 MinIO 时不应该校验野生存储配置的问题
- **修复** `mcamel-minio` 修复 创建 Bucket 偶尔会失败
- **升级** `mcamel-minio` [golang.org/x/net](https://golang.org/x/net) 到 v0.7.0

**2023-02-23****v0.3.0**

- **新增** `mcamel-minio helm-docs` 模板文件
- **新增** `mcamel-minio` 应用商店中的 Operator 只能安装在 `mcamel-system`
- **新增** `mcamel-minio` 支持 cloud shell
- **新增** `mcamel-minio` 支持导航栏单独注册
- **新增** `mcamel-minio` 支持查看日志
- **新增** `mcamel-minio` Operator 对接 chart-syncer
- **修复** `mcamel-minio` 实例名太长导致自定义资源无法创建的问题
- **修复** `mcamel-minio` 工作空间 Editor 用户无法查看实例密码
- **升级** `mcamel-minio` 升级离线镜像检测脚本
- **新增** 日志查看操作说明，支持自定义查询、导出等功能

**2022-12-25****v0.2.0**

- **新增** `mcamel-minio` NodePort 端口冲突提前检测
- **新增** `mcamel-minio` 节点亲和性配置
- **修复** `mcamel-minio` 修复单实例的时候，状态显示异常的问题
- **修复** `mcamel-minio` 创建实例时未校验名字
- **优化** `mcamel-minio` 取消认证信息输入框

**2022-11-28****v0.1.4**

- **改进** 更新获取前端的接口，增加 `sc` 列表是否可以扩容
- **改进** 密码校验调整为 MCamel 中等密码强度
- **新增** 创建 MinIO 集群时配置 Bucket
- **新增** 返回列表或者详情时的公共字段
- **新增** 返回告警列表
- **新增** 校验 Service 注释
- **修复** 创建 MinIO 时，密码校验从 `between` 调整为 `length`
- **改进** 完善优化复制功能
- **改进** 实例详情 - 访问设置，移除 集群 IPv4
- **改进** 中间件密码校验难度调整
- **新增** minio 支持创建的时候内置 BUCKET 创建

- **新增** 对接告警能力
- **新增** 新增判断 sc 是否支持扩容并提前提示功能
- **优化** 优化安装环境检测的提示逻辑&调整其样式
- **优化** 中间件样式走查优化

## 2022-11-08

### v0.1.2

- **新增** 获取用户列表接口
- **新增** minio 实例创建
- **新增** minio 实例的修改
- **新增** minio 实例的删除
- **新增** minio 实例的配置修改
- **新增** minio 实例支持 nodeport 的 svc
- **新增** minio 实例的监控数据导出
- **新增** minio 实例的监控查看
- **新增** 多租户全局管理的对接
- **新增** mcamel-minio-ui 的创建/列表/修改/删除/查看
- **新增** APIServer/UI 支持 mtls
- **修复** 单例模式下，只有一个 pod，修复 grafana 无法获取数据问题
- **优化** 完善了计算器功能

# 创建 MinIO 实例

1. 从左侧导航栏选择 **Minio 存储**。

选择 minio 存储

选择 minio 存储

2. 可以点击列表右上角的 **新建实例** 按钮。

如果是首次使用，需要[先选择工作空间](#)，然后点击 **立即部署** 创建 MinIO 实例。

点击新建实例

点击新建实例

3. 参考下方信息填写实例基本信息，然后点击 **下一步**。

**!!! note**

- 实例名称、所在集群/命名空间在实例创建之后不可修改
- 注意查看输入框下方的填写要求，输入符合要求的内容
- 如未通过安装环境检测，需要根据提示安装组件之后方可进行下一步

基本信息

基本信息

4. 参考下方信息填写配置规格，然后点击 **下一步**。

- 部署模式在实例创建之后不可更改
- 生产模式下建议采用高可用部署模式
- 高可用模式下需要至少 4 个副本
- 存储类：所选的存储类应有足够的可用资源，否则会因资源不足导致实例创

建失败

- 存储容量：每个磁盘具有多少容量。实例创建之后不可调低
- 每副本磁盘数：为每个副本提供多少个次盘。实例创建之后不可调低

**!!! warning**

MinIO 实例创建成功后，不支持扩容副本和修改磁盘用量，请谨慎配置资源。

**配置规格****配置规格**

5. 参考下方信息填写服务设置，点击 **下一步**。

- 集群内访问：只能在同一集群内部访问服务
- 节点端口：通过节点的 IP 和静态端口访问服务，支持从集群外部访问服务
- 负载均衡器：使用云服务提供商的负载均衡器使得服务可以公开访问
- 负载均衡器/外部流量策略：规定服务将外部流量路由到节点本地还是集群范围内的断点
  - Cluster：流量可以转发到集群中其他节点上的 Pod
  - Local：流量只能转发到本节点上的 Pod
- 控制台账号：访问此新建实例时需要用到的用户名、密码

**访问模式****访问模式**

??? note “点击查看高级配置说明”

- Bucket 名称：在此实例下新建一个存储桶，设置新建存储桶的名称
- 调度策略/条件：设置 Pod 调度的节点亲和性，可参考 Kubernetes 官方文档[节点亲和性](<https://kubernetes.io/zh-cn/docs/concepts/scheduling-eviction/assign-pod-node/#affinity-and-anti-affinity>)
  - 尽量满足：先尝试调度到满足规则的节点。如果找不到匹配的节点，也会执行 Pod 调度
  - 必须满足：只有找到满足规则的节点时才执行 Pod 调度
- 调度策略/权重：为满足每条调度策略的节点设置权重，优选使用权重高的策略。取值范围 1 到 100
- 调度策略/选择器
  - In：节点必须包含所选的标签，并且该标签的取值必须 \*\*属于\*\* 某个取值集

合。多个值用 `_; _` 隔开

- `NotIn`: 节点必须包含所选的标签，并且该标签的取值必须 \*\*不属于\*\* 某个取值集合。多个值用 `_; _` 隔开

- `Exists`: 节点包含某个标签即可，不关注标签的具体取值

- `DoesNotExist`: 节点不包含某个标签，不关注标签的具体取值

- `Gt`: 节点必须包含某个标签，并且标签的取值必须大于某个整数

- `Lt`: 节点必须包含某个标签，并且标签的取值必须小于某个整数

![访问模式](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/minio/images/create09.png)

6. 确认实例配置信息无误，点击 **确定** 完成创建。

**点击 确定**

**点击 确定**

7. 返回实例列表页查看实例是否创建成功。

创建中的实例状态为 **未就绪**，等所有相关容器成功启动之后状态变为 **运行中**。

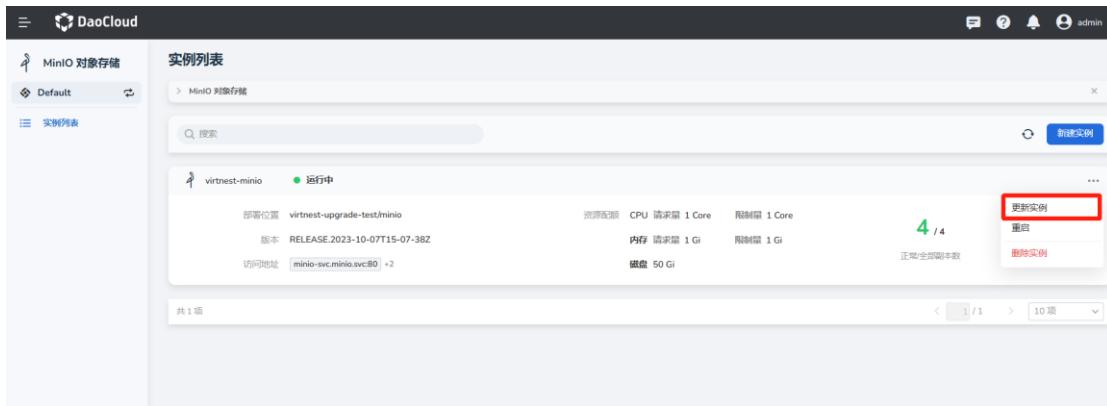
状态  
状态

## 更新、删除 MinIO 实例

### 更新 MinIO 实例

如果想要更新或修改 MinIO 的资源配置，可以执行如下操作：

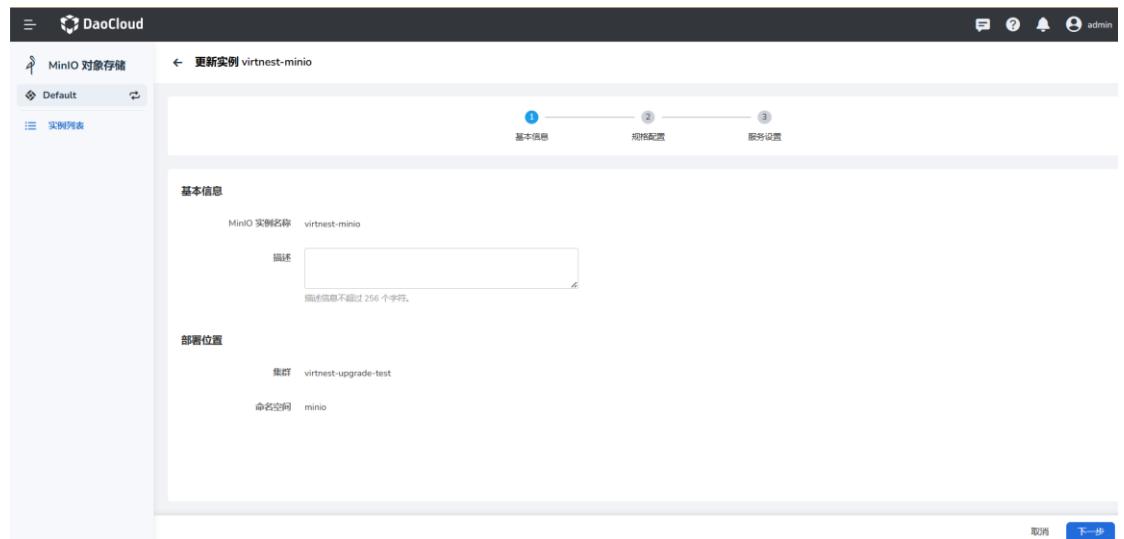
1. 在实例列表中，点击右侧的 `...` 按钮，在弹出菜单中选择 **更新实例**。



选择 **更新实例**

2. 修改基本信息，然后点击 **下一步**。

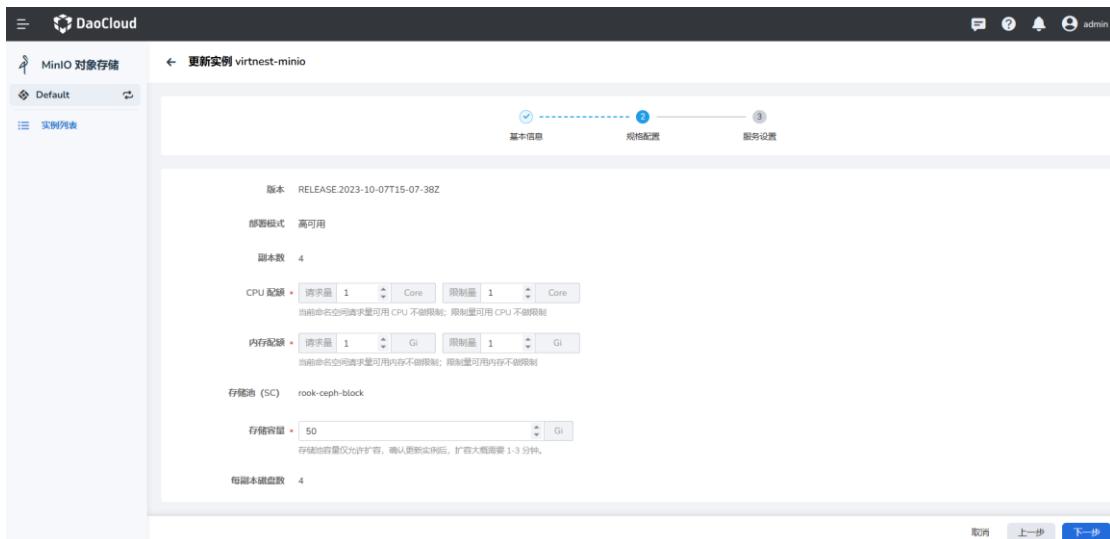
- 仅支持修改描述信息
- 实例名称、部署位置不可修改



### 基本信息

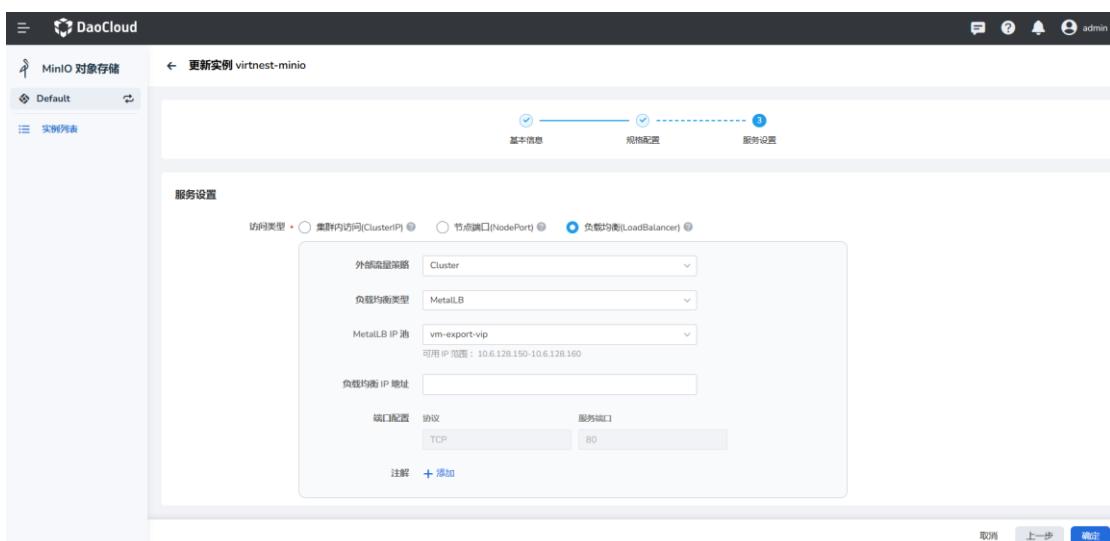
3. 修改规格配置，然后点击 **下一步**。

- 仅支持修改：CPU 配额、内存配额和储存容量
- 不可修改：版本、部署模式、副本数、存储池、每副本磁盘数



### 规格配置

4. 修改服务设置，然后点击 确定。



### 服务设置

5. 返回实例列表，屏幕右上角将显示消息：更新实例成功。

成功

成功

## 删除 MinIO 实例

如果想要删除一个实例，可以执行如下操作：

1. 在实例列表中，点击右侧的 ... 按钮，在弹出菜单中选择 **删除实例**。

选择删除实例

选择删除实例

2. 在弹窗中输入该实例列表的名称，确认无误后，点击 **删除** 按钮。

!!! warning

删除实例后，该实例相关的所有消息也会被全部删除，请谨慎操作。

点击删除

点击删除

## 实例监控、日志查看与配置参数

### 实例监控

MinIO 内置了 Prometheus 和 Grafana 监控模块。

1. 在实例列表页面中，找到想要查看监控信息的实例，点击该实例的名称。

点击某个名称

点击某个名称

2. 在左侧导航栏点击 **实例监控**。

点击实例监控

点击实例监控

!!! note

如果提示监控组件异常，请按提示[安装 insight-agent 插件](../../../../insight/quickstart/install/install-agent.md)。

3. 查看实例的监控信息。点击红框里的信息符号可查看每个指标的含义说明。

点击实例监控

点击实例监控

## 日志查看

通过访问每个 MinIO 的实例详情页面，可以支持查看 MinIO 的日志。

1. 在 MinIO 实例列表中，找到想要查看日志的实例，点击实例名称进入详情页面。

image

image

2. 在左侧菜单栏点击 **日志查看**。

image

image

3. 根据需要调整日志查询的时间范围和刷新周期，具体可参考[日志查询](#)。

image

image

!!! note “常用操作”

- \* 自定义日志查询时间范围：在日志页面右上角可以切换日志的查询时间范围
- \* 通过关键字检索日志：在左侧 搜索框 下输入关键字即可查询带有特定内容的日志
- \* 查看日志的时间分布情况：通过柱状图查看在某一时间范围内的日志数量
- \* 查看日志的上下文：在日志最右侧点击 查看上下文 即可
- \* 导出日志：在柱状图下方点击 下载 即可

![image](https://docs.daocloud.io/daocloud-docs-images/docs/middleware/minio/images/log03.png)

## 配置参数

MinIO 内置了参数配置 UI 界面。

1. 在 MinIO 实例列表中，找到想要配置参数的 MinIO 实例，点击实例名称。

点击某个名称

点击某个名称

2. 在左侧导航栏，点击 **配置参数**。

点击配置参数

点击配置参数

3. 打开 **编辑配置**，对 MinIO 的各项参数进行添加、删除、修改。

### 配置参数

#### 配置参数

4. 点击 **保存**，会导致 MinIO 重启。

## 查看实例详情

本节说明如何查看 MinIO 实例。

1. 在实例列表页面中，找到想要查看详情的实例，点击实例名称。

### 点击实例名称

#### 点击实例名称

2. 查看基本信息、访问设置、资源配额和 Pod 列表等信息。

在当前页面除了查看基本信息外，还支持下列操作：

- 开启云端控制台通过命令行对实例进行相关操作
- 更新或删除实例
- 点击访问地址可以在 Web 界面中打开控制台并使用右侧的用户名和密码进行登录

### 行登录

- 可以查看或创建告警规则
- 点击容器组名称可以跳转到容器管理模块查看该容器组的详细情况

### 查看

#### 查看

# MinIO 的身份管理

DCE 5.0 提供的 MinIO 服务自带网页控制台（ Web Console ）。了解 MinIO 的身份管理

（ identity management ）有助于快速了解如何在 MinIO 内安全有效地管理子账号。

本文简单介绍 MinIO 的身份管理规则，更多详细说明可参考 [MinIO 的官方文档](#)。

## 用户

默认情况下，MinIO 使用内置的 Identity Provider ( IDP ) 来完成身份管理。除了 IDP ，还

支持第三方 [OIDC](#) 和 [LDAP](#) 的方式。

用户由一对 `username` 和 `password` 组成。在 MinIO 的语境中，`username` 又被称为 `access key` （注意与后面 `service account` 层级的 `access key` 区分开来），`password` 又称为 `secret key` 。

### root 用户

在启动 MinIO 时，可以通过环境变量的方式设置 MinIO 集群中 root 用户的账号密码，

分别是以下两个变量：

- `MINIO_ROOT_USER`
- `MINIO_ROOT_PASSWORD`

root 用户拥有所有资源的所有操作权限。

注意：如果要变更 root 用户，需要重启 MinIO 集群中所有的节点。

### 普通用户

支持通过三种方式创建普通用户：

- Web Console，在 UI 界面中通过表单进行创建

- mc , 使用 CLI 命令行创建
- Operator CR , 使用 CR 进行创建

## Console 创建

1. 在 DCE 5.0 的 MinIO 实例详情页面 , 点击访问地址 , 使用右侧的用户名和密码即可登录该实例的 Console 控制台。

登录 Console

登录 Console

2. 登录 Console 控制台之后根据下图所示 , 创建用户。

通过 Console 创建普通用户

通过 Console 创建普通用户

## mc 创建

需要事先[安装 mc 命令](#) , 并配置连接到 MinIO 实例

创建用户 :

**ALIAS** 指 MinIO 实例的别名

```
mc admin user add ALIAS ACCESSKEY SECRETKEY
```

授予权限 :

**USERNAME** 指 MinIO 用户的用户名 , 即 **ACCESSKEY**

```
mc admin policy set ALIAS readwrite user=USERNAME
```

## operator CR 创建

如果是通过 cr 安装 MinIO , 也可以通过 users 字段来指定普通用户的 secret :

```

type TenantSpec struct {
 ...
 ...
 ...
 // *Optional* +
 //
 // An array of https://kubernetes.io/docs/concepts/configuration/secret/ [Kubernetes opaque secrets] to use for generating MinIO users during tenant provisioning. +
 //
 // Each element in the array is an object consisting of a key-value pair __name: <string>__ , where the __<string>__ references an opaque Kubernetes secret. +
 //
 // Each referenced Kubernetes secret must include the following fields: +
 //
 // * __CONSOLE_ACCESS_KEY__ - The "Username" for the MinIO user +
 //
 // * __CONSOLE_SECRET_KEY__ - The "Password" for the MinIO user +
 //
 // The Operator creates each user with the __consoleAdmin__ policy by default. You can change the assigned policy after the Tenant starts. +
 // +optional
 Users []*corev1.LocalObjectReference __json:"users,omitempty"__
 ...
 ...
 ...
}

```

## 服务账号

服务账号 (Service Account) 通常使用用户登录 console 或者通过 mc 命令对 MinIO 进行管理操作。但如果应用程序需要访问 MinIO，则通常使用 Service Account (这是比较正式的叫法，某些上下文中也称之为 access key )。

一个用户可以创建多个 Service Account。

**注意：**无法通过 Service Account 登录 MinIO console，这也是它与用户最大的不同之处。

## Console 创建

通过 console 创建 service account

通过 console 创建 service account

## mc 命令创建

```
mc [GLOBALFLAGS] admin user svcacct add \
 [--access-key] \
 [--secret-key] \
 [--policy] \
 ALIAS \
 USER
```

有关 MinIO 用户的详细说明，可参考 [User Management](#)

## 用户组

用户组，顾名思义即多个用户形成的集合。通过用户组合结合授权策略可以批量管理一组用户的权限。通过授权策略可以为用户组分配资源权限，该组中的用户会继承用户组的资源权限。

MinIO 用户的权限分为两部分：用户原本具有的权限 + 从所在用户组继承而来的权限。在 MinIO 语境中，用户仅具有其明确被授予或从用户组继承而来的授权。如果用户没有明确获得（被直接授予或继承）某一资源的权限，则无法访问该资源。

有关 MinIO 用户组的详细说明，可参考 [Group Management](#)

## 授权策略

MinIO 使用基于策略的访问控制 (PBAC) 来管理用户对哪些资源具有哪些权限。每条策略通过规定一些动作或条件来限制用户和用户组具有的权限。

## 内置策略

MinIO 内置了四种策略可以直接分配给用户或用户组。为用户/用户组授权时需要使用 `mc`

`admin policy set` 命令，具体可参考 [mc admin policy](#)

- `readonly`：对 MinIO 副本中的所有存储桶和存储对象具有 **只读** 权限
- `readwrite`：对 MinIO 副本中的所有存储桶和存储对象具有 **读写** 的权限
- `diagnostics`：对 MinIO 副本具有 **诊断** 权限
- `writeonly`：对 MinIO 副本中的所有存储桶和存储对象具有 **只写** 权限

## 策略文件示例

MinIO 授权策略文件的模式和[亚马逊云 IAM Policy](#) 相同。

```
{
 "Version" : "2012-10-17",
 "Statement" : [
 {
 "Effect" : "Allow",
 "Action" : ["s3:<ActionName>", ...],
 "Resource" : "arn:aws:s3:::*",
 "Condition" : { ... }
 },
 {
 "Effect" : "Deny",
 "Action" : ["s3:<ActionName>", ...],
 "Resource" : "arn:aws:s3:::*",
 "Condition" : { ... }
 }
]
}
```

有关 MinIO 授权策略的详细说明，可参考 [Policy Management](#)

# MinIO 灾备方案

## 功能介绍

MinIO 客户端命令 `mc replication` 用于配置在 MinIO 服务器之间存储桶的复制。Site Replication（站点复制）是 MinIO 中用于实现跨多个站点的数据复制的功能。这个功能可以用来实现异地灾备，保证数据在多个物理位置的多个 MinIO 集群间实时同步。

Site Replication 功能要求 MinIO 部署在分布式模式下，并且每个站点都有独立的 MinIO 集群。此外，各个集群之间需要有足够的网络带宽来支持数据复制。

## 使用命令行进行灾备

以下是使用 MinIO 命令行进行异地灾备的步骤：

1. 准备两套 MinIO 集群，执行以下两条命令在 MinIO 客户端中添加需要进行灾备的

### MinIO。服务器

```
mc alias set minio1 http://host1:9000 user password
mc alias set minio2 http://host2:9000 user password
```

- **minio1**：用户为 MinIO 服务端点设置的别名，
- **http://host1:9000**：是 MinIO 服务的 URL
- **user**：这是连接 MinIO 服务所需的用户名。
- **password**：这是连接 MinIO 服务所需的密码。

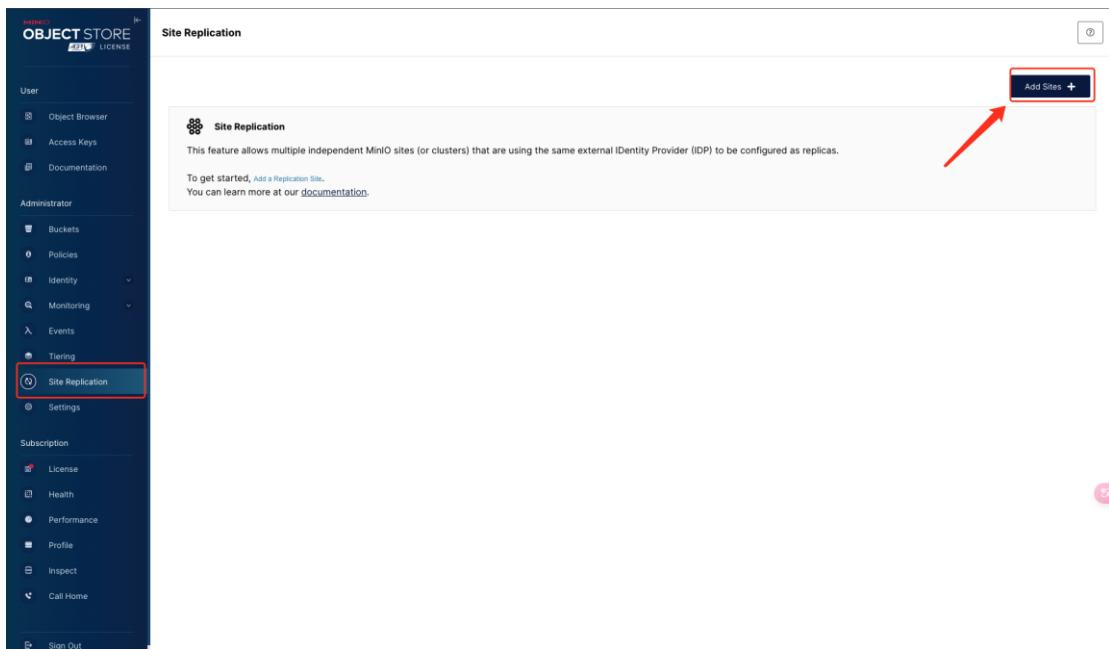
2. 执行以下命令在 `minio1` 服务上添加一个复制规则，以便自动将数据复制到别名为 `minio2` 的目标 MinIO 服务。

```
mc admin replicate add minio1 minio2
```

# 使用控制台进行灾备

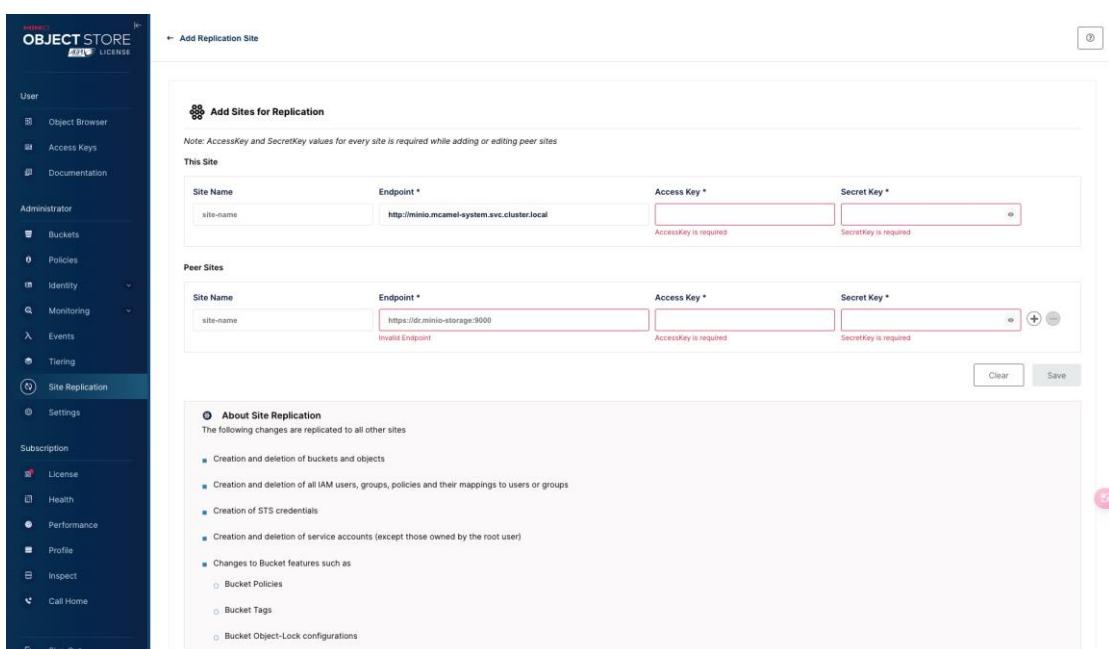
MinIO 的控制台 ( Console ) 提供了一个图形化界面，通过它可以管理 MinIO 对象存储服务，以下为执行步骤：

1. 登录待备份集群的页面控制台，进入左侧导航栏的 Site Replication，点击 **Add Sites**



minio 灾备

2. 填写当前 MinIO 集群的地址以及目标 MinIO 集群的地址，完成后点击 **Save**



## minio 灾备