Apusic 金蝶天燕

生成时间：2026-04-01

用户手册

金蝶Apusic云原生中间件平台

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 产品介绍

金蝶Apusic云原生中间件平台(Apusic Cloud Middleware Plantform)是基于K8S-Operator技术构建的一站式云中间件管理平台。 满足用户对中间件部署、变更、监控、运维等全生命周的管理需求，帮助以企业以最佳实践的方式来使用和管理中间件。充分利用云中间件的快速部署、弹性伸缩、动态配置、高可用高可靠的特性，来为企业应用提供现代化的应用支撑能力。

2 范围和读者

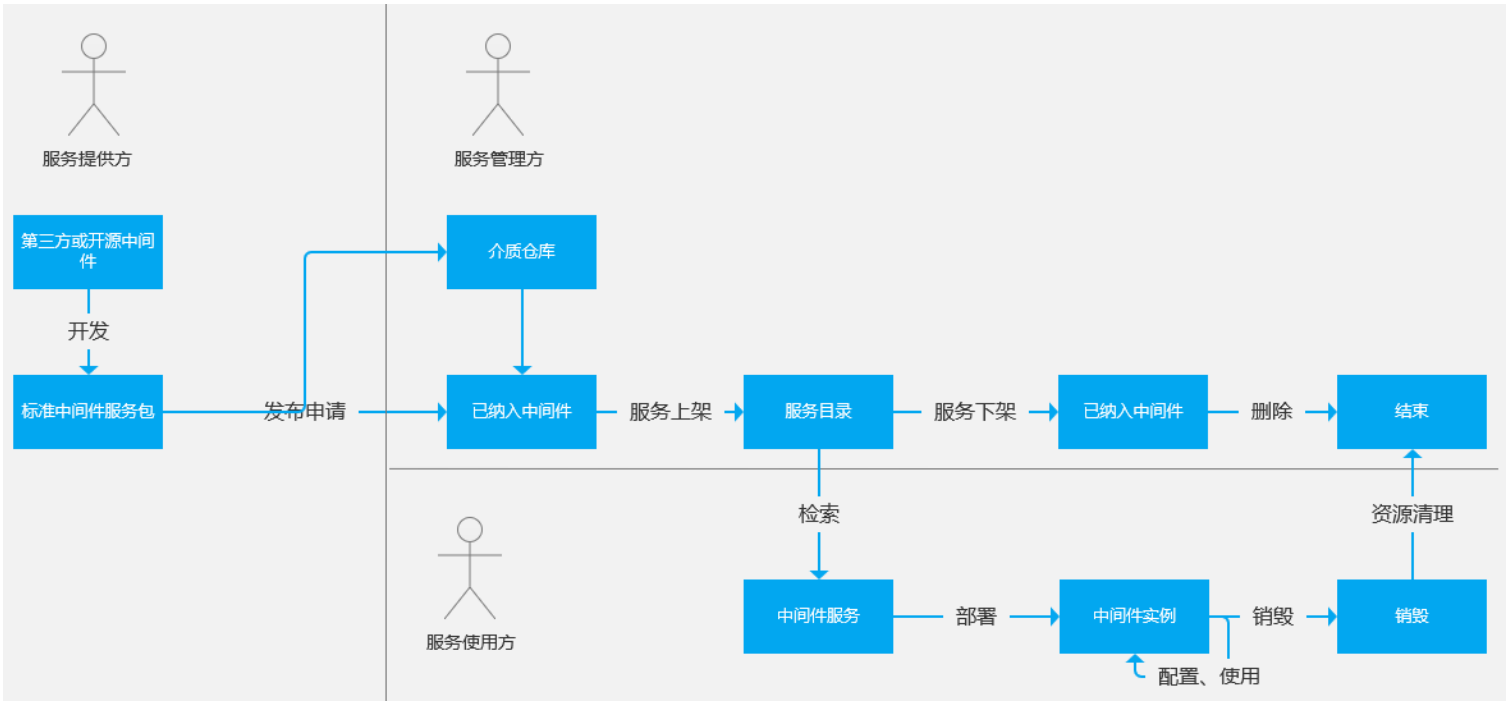
本手册介绍ACMP产品的使用详细说明，适用于ACMP产品的用户，ACMP产品技术顾问，ACMP产品维护人员，以及希望学习了解ACMP产品的相关人员。

3 用户与角色

- 服务使用者：使用平台中的服务能力部署自己所需的产品服务，并通过平台管理其生命周期。
- 服务管理者：验证、测试服务提供者提供的软件能力和管理其版本。为服务使用者分配资源，并帮助其更好的使用平台内的服务。
- 服务开发者：提供基础软件能力的团队或厂商。在本平台中以服务包形式封装产品能力，并上传到平台。

4 生命周期

- 服务发布：服务开发者对中间件中间件进行封装后，形成标准服务包，由服务管理者上线到服务目录。
- 服务使用：用户从服务目录中检索到服务后，可以进行服务部署生成服务实例，并对服务实例进行配置调整。当用户不在使用后，可以销毁服务实例。
- 服务下线：当没有服务使用者继续使用服务时，服务管理者可以将服务下架。服务目录中将不再能够检索到该服务，当确认所有实例都已销毁后和资源清理后，可以删除该服务。



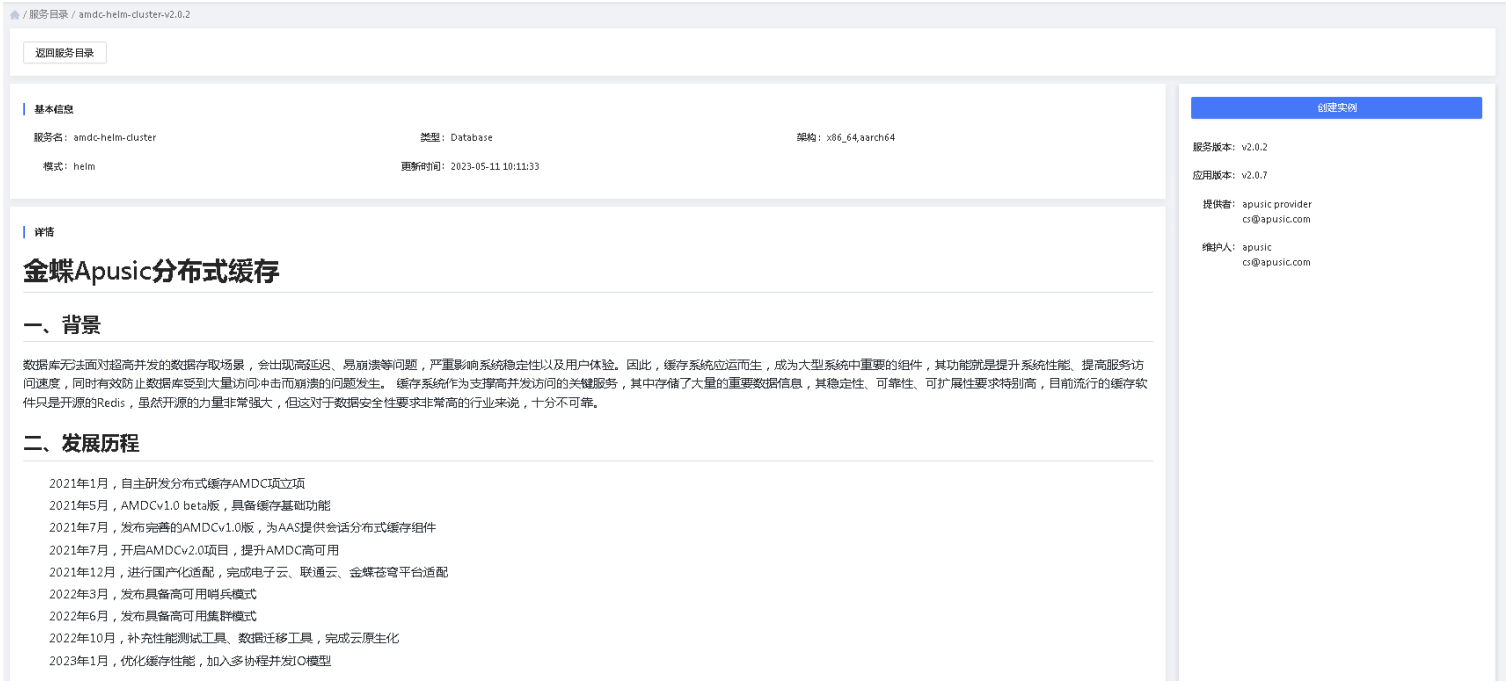
5 服务使用者指南

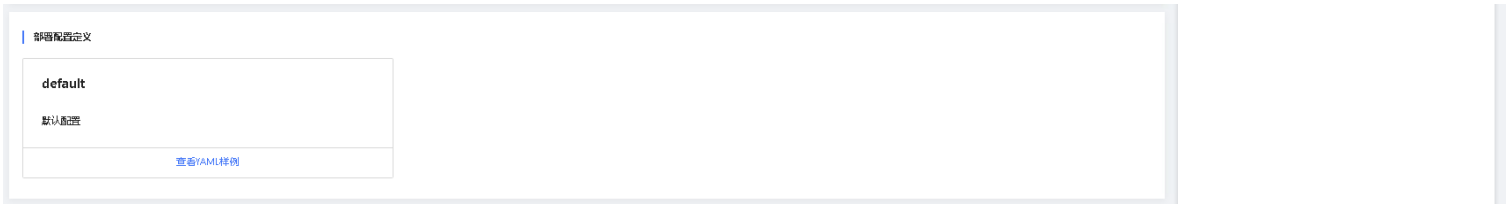
5.1 服务检索

服务目录为普通用户提供检索服务的能力，用户可以根据多种条件对服务进行检索。服务目录提供服务的详细介绍信息，使用户可以深入的了解每个服务差异，一边更好的选择合适的服务。

1.用户可以根据关键词、类型、架构、部署模式快速找到自己所需要的服务。服务目录中包含各类开源与商业产品，可根据自己的需要进行选择。

2.当用户选定一个服务时，可以点击该服务查看其服务详情。



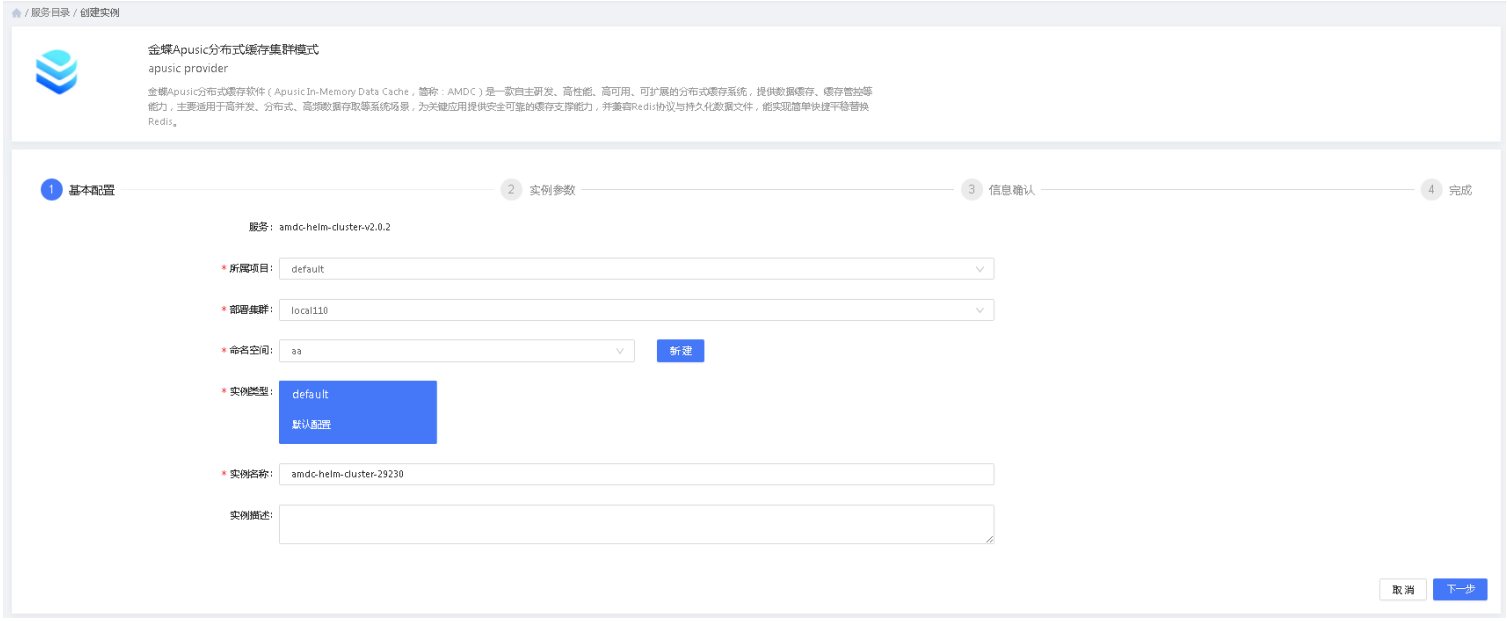


- 创建实例：用户可以通过该按钮进入服务实例的安装过程
- 服务名：服务名全局唯一，是服务的唯一标识。
- 类型：服务的所属业务类型
- 架构：该服务是否支持x86_64或aarch64。如果多包含标识该服务同时支持两种类型
- 模式：说明该服务以operator还是helm进行部署。
- 更新时间：该服务的发布时间，如果该服务后续有更新，则显示最后一次的发布时间
- 详情：服务提供商提供的服务详细描述信息，不同服务的描述信息格式可能不一致。
- 服务版本：该服务的具体版本
- 应用版本：该服务部署后安装的应用版本，同一服务版本可以安装多个应用版本的应用。但这里一般显示最大的应用版本。
- 提供者：该服务的提供人
- 维护人：该服务的维护人
- 部署配置定义：部分服务提供多种配置模式，用户可以点击"查看YAML样例"查看对应模式的具体配置。

5.2 服务实例部署

用户在服务详情中，点击"创建服务"，在进入对应服务的创建向导

1.步骤一：基本配置



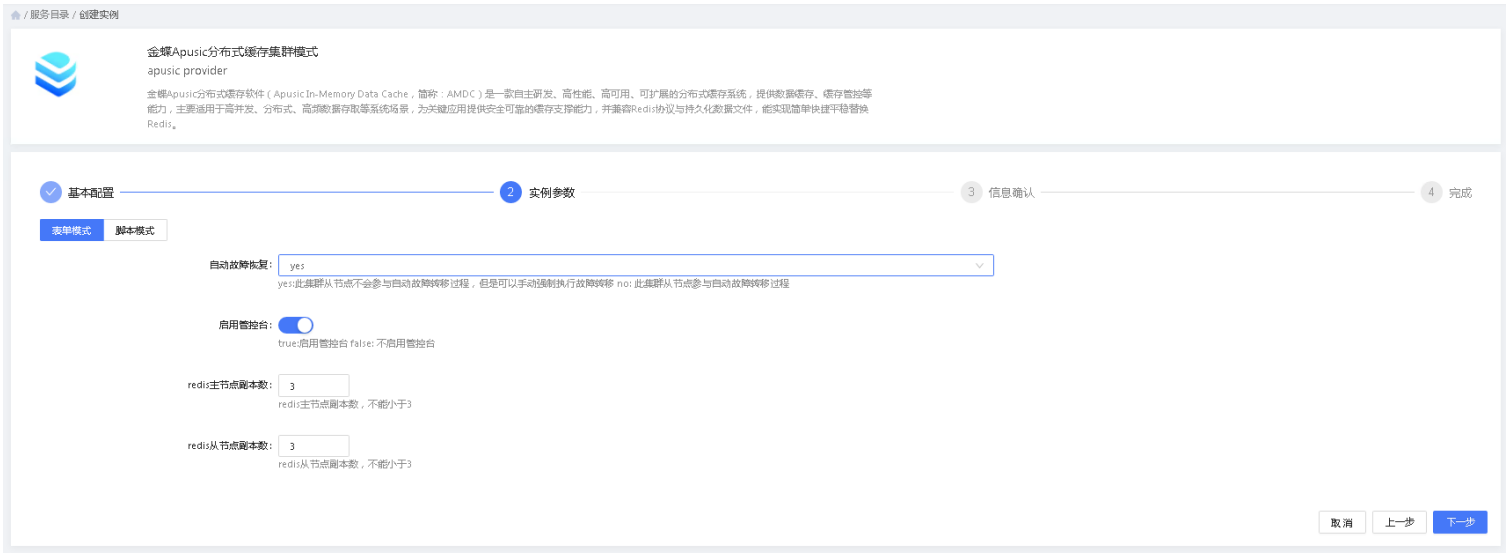
- 所属项目：该服务实例创建后所属的项目，该项目有权限的人员也可以对服务实例进行管理
- 部署集群：选择该部署需要部署的目标K8S集群
- 命名空间：需要部署服务的目标集群的命名空间
- 实例类型：若服务有多种配置类型，则此处会显示对应的类型及类型描述。用户可以根据需要选择合适自己场景的类型进行部署。
- 实例名称：实例名称是服务实例的唯一标识，不允许重复。
- 实例描述：可选，实例的描述信息。

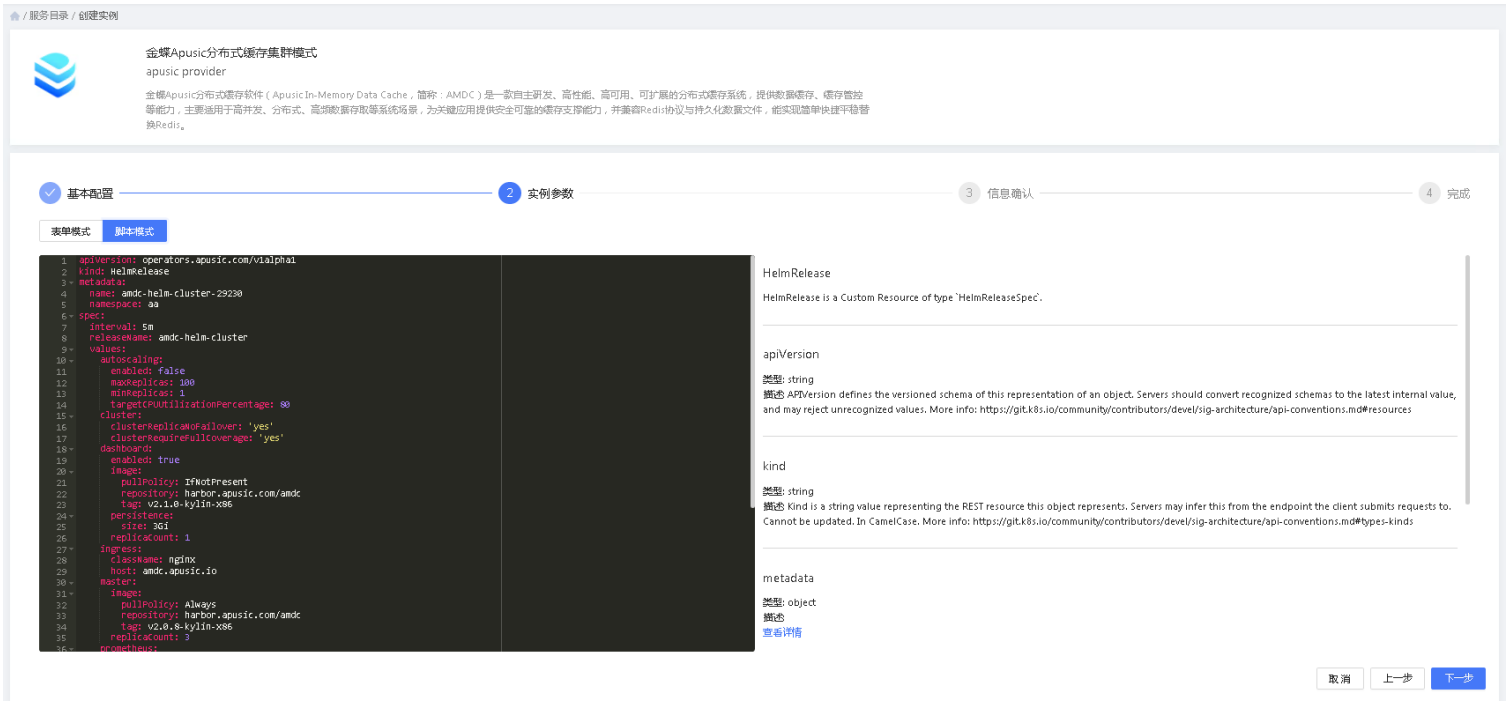
2.步骤二：实例参数

实例参数分为表单创建和脚本创建两种，"表单创建"提供带有指引说明的可视化配置模式，"脚本创建"则提供了更为直接的脚本配置模式。

用户可以在两种配置间无缝切换,两种配置模式中配置的参数值都会自动同步到另外一种模式中。

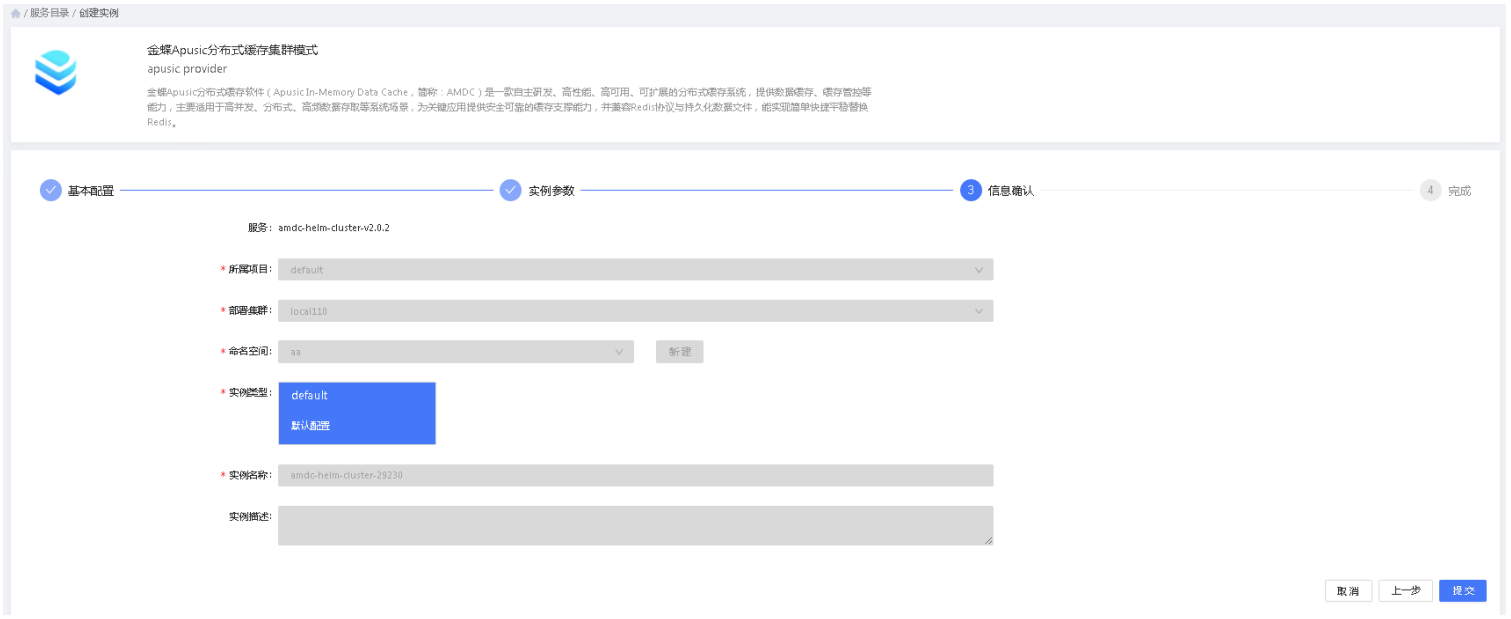
不同的中间件采用了不同的配置规则，故不同的中间件配置信息会有所差异，需要用户根据需要进行配置。





3.步骤三：信息确认

用户配置完成后，可以查看配置信息是否正确，若不正确还可返回修改。若正确，则可提交配置到平台进行部署。



4.步骤四：完成

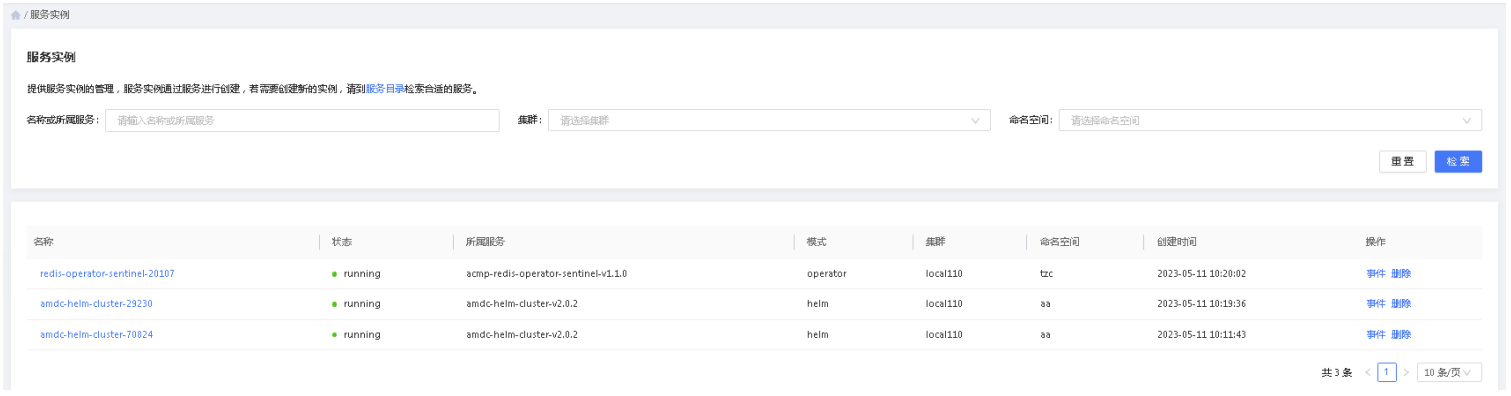
进入该步骤，标志配置信息已经通过平台的校验，并开始进行下发执行。用户可以选择"服务实例"查看实例安装的进度和状态。用户也可选择"服务目录"，回到服务目录继续挑选想要部署的实例。

5.3 服务实例管理

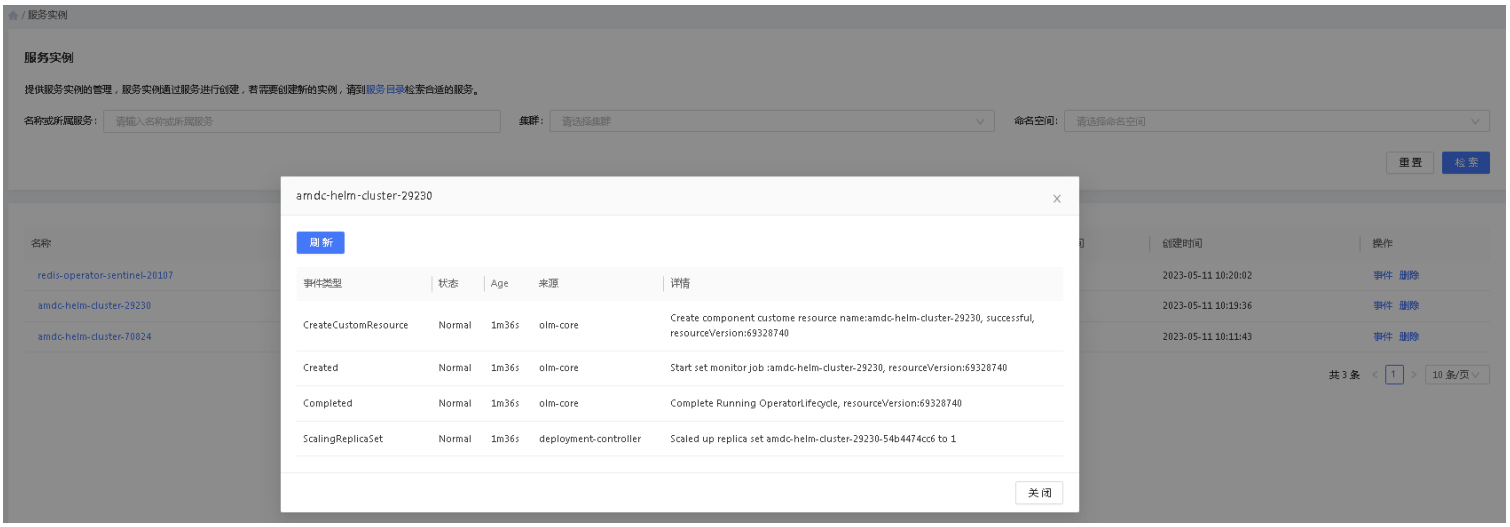
服务实例提供具体实例的管理的功能，用户可以查看信息、监控状态、进行运维操作以及删除不再需要的实例。

5.3.1 服务实例列表

服务实例列表提供用户有权限的所有实例信息查看。用户可以根据名称、所属服务名、部署集群和命名空间对服务实例进行检索。用户可以在列表中删除不再需要的服务实例，服务实例删除后，配置信息和数据可能丢失，需谨慎操作。

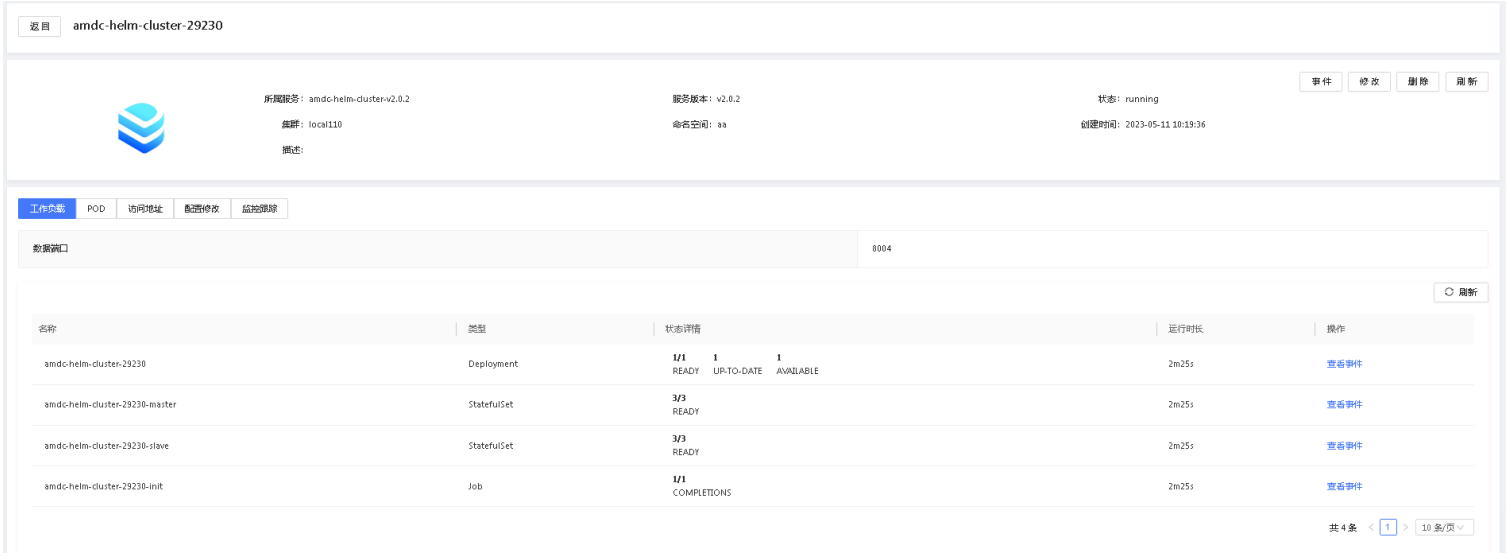


用户在执行服务部署后，可以查看对应的服务实例安装部署过程中所执行的事件，用户可以点击"事件"按钮来查看执行的详细情况，尤其是服务部署失败或者挂起的时候。



5.3.2 实例状态

在服务实例列表中点击服务实例名称，即可进入服务详情，查看该服务的详细信息。



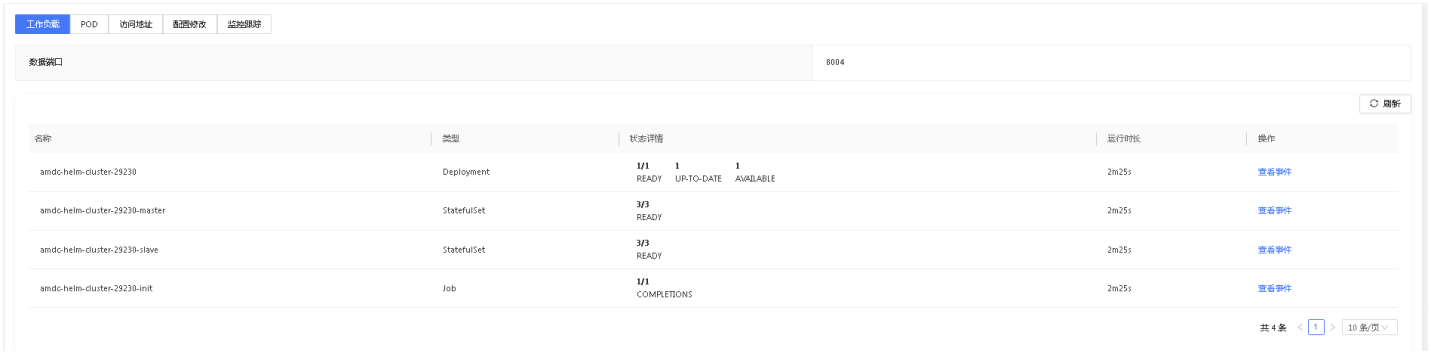
- 事件：提供服务近期的运行事件。
- 修改：提供服务实例重新部署能力，当服务实例配置后，会按照新重新进行部署。
- 删除：删除当前实例。
- 工作负载页签：提供实例所有工作负载状态情况，可以根据负载情况快速了解实例部件的状态。
- POD页签：提供服务实例包含的所有POD和容器的状态信息。
- 访问地址页签：提供服务实例暴露的服务地址和端口。
- 配置修改页签：提供服务实例的配置信息的查看和修改能力。
- 监控跟踪页签：提供基本的服务监控能力。

5.3.3 工作负载

工作负载是Kubernetes提供的控制器实现，能够帮助用户管理期望状态，自动调度底层Pod和容器为用户期望的状态和数量。
工作负载有多种类型，每种类型都有自己的状态：

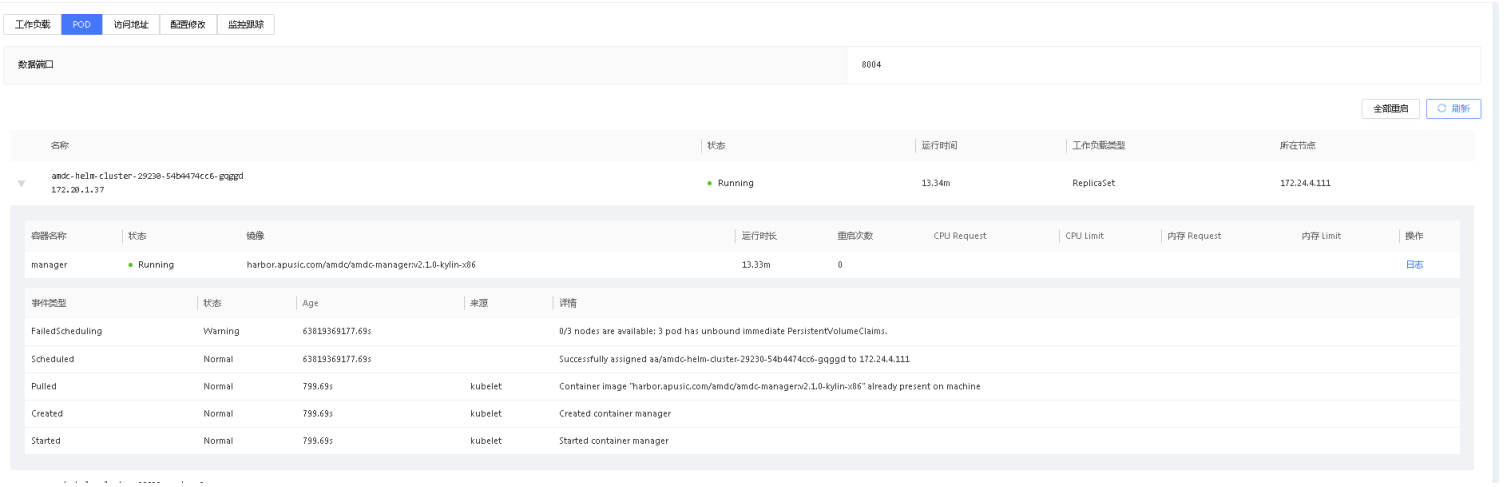
- Doployment:无状态控制器
- Statefulset:有状态控制器
- DaemonSet:守护进程控制器
- Job:一次性任务控制器

- Cronjob:定时任务控制器



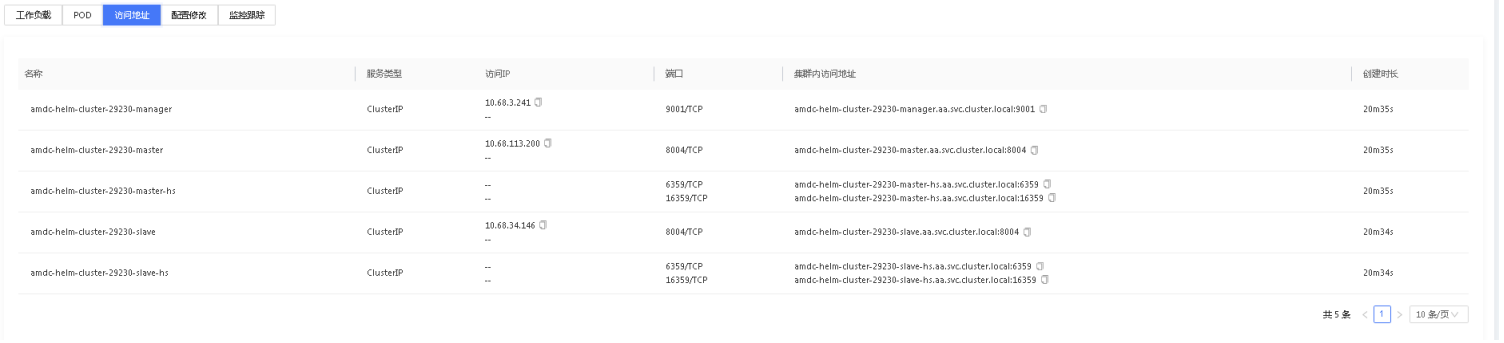
5.3.4 Pod

用户可以通过Pod页签查看服务实例的Pod和容器运行情况。一个Pod下可能包含多个容器，可以通过点击Pod列表左侧的下拉列表查看Pod下容器的具体信息。 用户可以点击日志，在弹出窗口中查看当前容器的运行日志。



5.3.5 访问地址

访问地址展示了该服务实例所有的端口暴露情况和对应的集群访问地址。



5.3.6 配置修改

用户可以通过配置修改功能查看和修改服务实例涉及到的所有配置信息。

5.3.7 监控跟踪

服务实例包含常见的监控图表，用于用户监控该实例的运行情况。

6 服务管理者指南

6.1 配置Kubernetes集群

用户可以通过"集群"功能，查看当前平台接入的K8S集群。

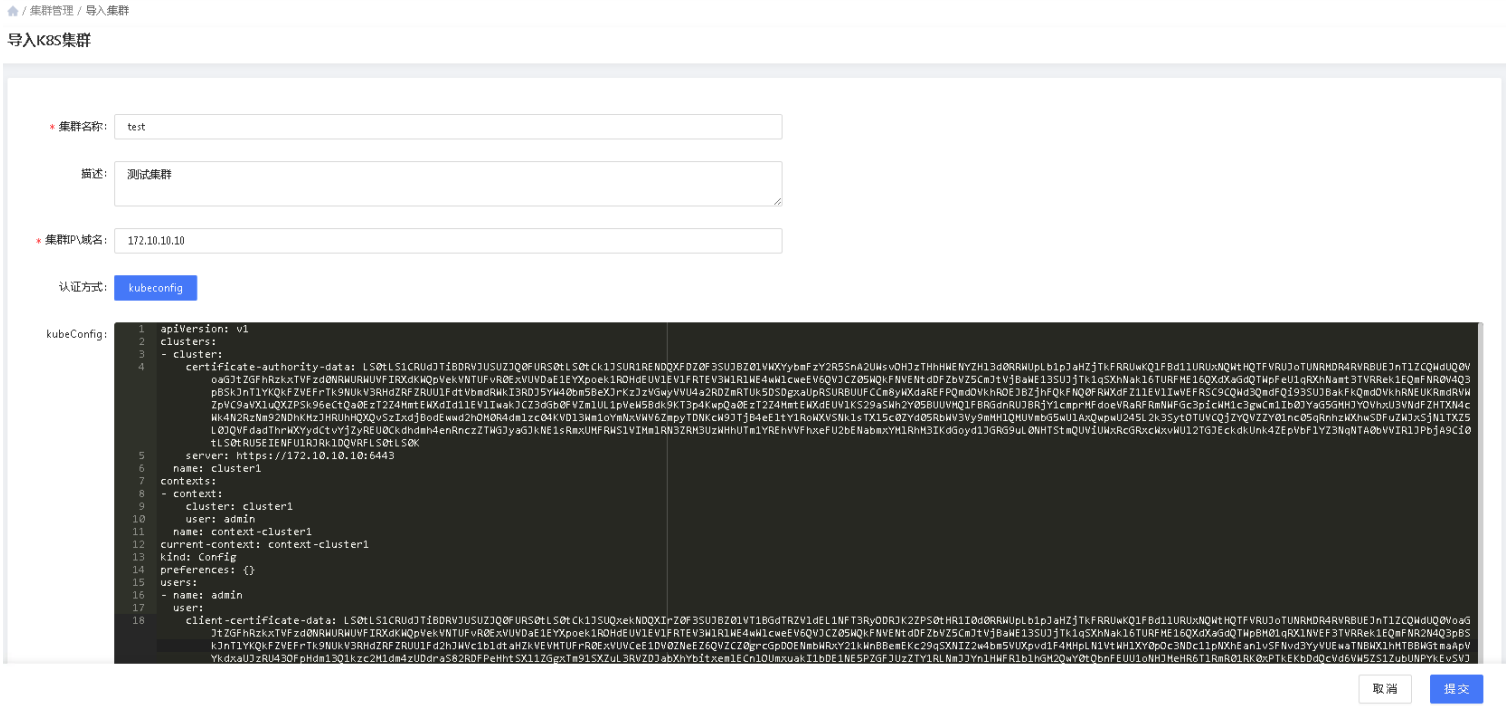
6.1.1 查看集群

local集群代表默认的控制面和数据面，可用于业务测试。

- 集群名称：集群的唯一标识
- 创建时间：集群被加入到平台管理的时间
- 集群版本：集群的K8S版本
- 负责人：当前集群的管理人
- 来源类型：标识集群是通过平台创建的还是外部创建后导入的
- 可以用节点数：当前集群下的节点数量
- CPU利用率：当前系统和业务占用的CPU核数
- 内存利用率：当前系统和业务占用的内存量（GB）

6.1.2 添加或修改集群

用户可以自行添加集群到平台中，在单集群模式下，系统只允许添加一个集群。
集群被添加后不允许随意修改。以免影响业务的正常运行。



- 集群名称：集群的唯一标识
- 描述：该集群的作用描述

- 集群IP\域名：描述该集群的默认接入地址
- kubeConfig：集群的接入配置信息

6.1.3 删除集群

当集群不再使用时，用户可以将集群连接进行删除。由于删除集群会影响正在运行的业务，故集群中还存在服务实例时，该集群不能被删除。

6.2 工作空间与配额管理

kubernetes通过命名空间来分割和管理资源，通过为命名空间添加配额来限制资源占用，从而避免个别业务对资源的过度使用。

6.2.1 查看命名空间

可以通过点击集群名称进入集群，查看该集群下的命名空间情况。

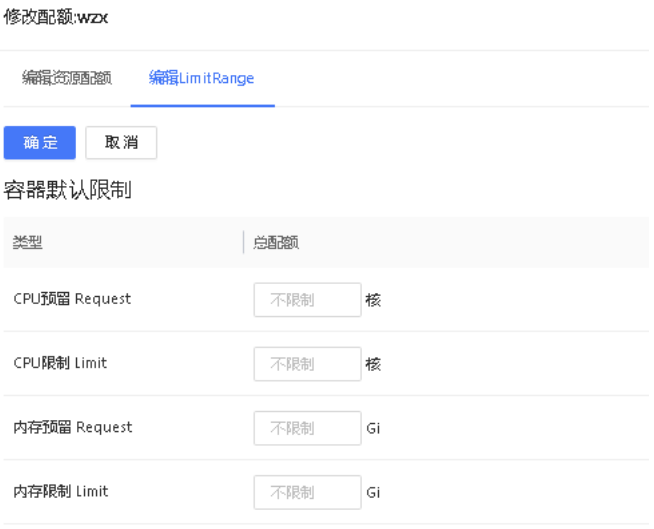
- 名称：命名空间唯一标识，不允许重复
- 描述：该命名空间的作用描述
- 状态：提供命名空间的状态情况。Active：正常 terminating：销毁中
- 标签：命名空间标签列表
- 存活时间：命名空间创建以来经过的时间
- 修改配额：为命名空间设置配额

6.2.2 新增命名空间

用户可以根据需要创建自己的命名空间

6.2.3 设置资源配额

为了避免工作空间内的应用过度占用资源，需要为工作空间设置资源上限，用户可以通过命名空间修改配额功能对其进行设置。
如需要了解相关概念，请查看Kubernetes官网相关章节： 限制范围(limit-range) 资源配额(resource-quotas)



- 设置容器默认限制范围

- 设置命名空间资源配额

6.2.4 删除命名空间

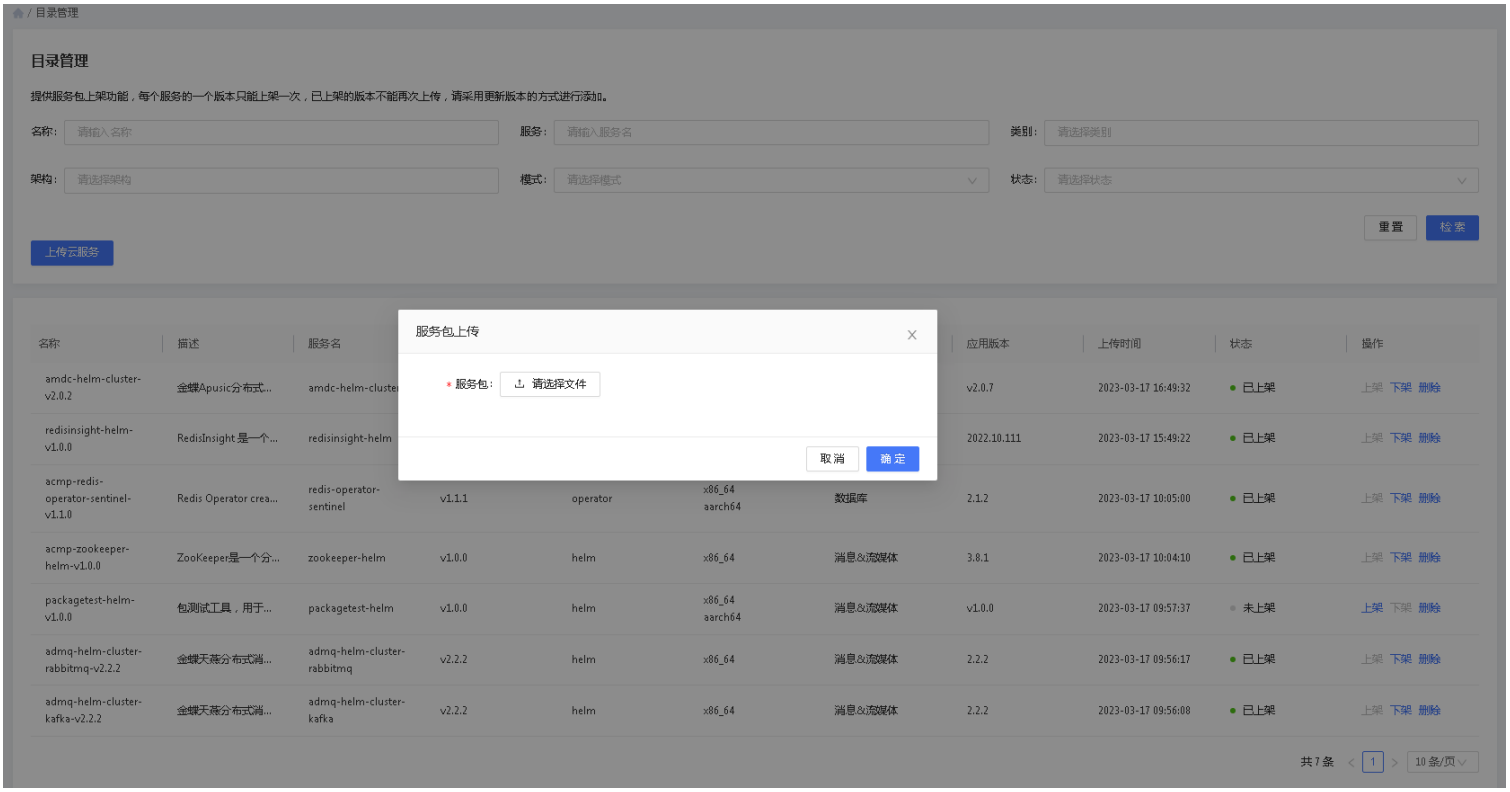
当命名空间不再使用时，可以删除空间。但命名空间删除后，将导致命名空间下的数据全部丢失，需谨慎操作。

6.3 服务管理与发布

服务发布提供了平台管理员将服务发布到平台中的能力，ACMP平台提供服务包制作规范，按照该服务规范开发的服务包，可以通过服务发布功能安装到平台中，使平台具备安装和使用该服务内中间件的能力。

6.3.1 服务包上传

进入平台侧服务发布菜单中，选择"上传云服务按钮"，选择对应的服务包进行上传。服务包上传后，暂时不能使用，需要由管理员进行上架后才能被其它用户所看到。
若服务包未能通过平台的合法性校验，则该服务包无法上传成功。



6.3.2 服务上架与下架

点击需要发布服务的"上架"按钮，即可完成服务上架。服务上架后，平台中的用户即可在服务目录中看到和使用该服务。
若该服务不再使用，则可点击需要取消服务的"下架"按钮，该服务则在服务目录中不再可见。

6.3.3 服务删除

当服务不再使用时，可以下架的服务进行删除。删除服务不会影响既有的服务，但可能会影响后续的服务升级。 故建议确认平台内不再有该服务的实例后，再进行删除。

7 服务开发者指南

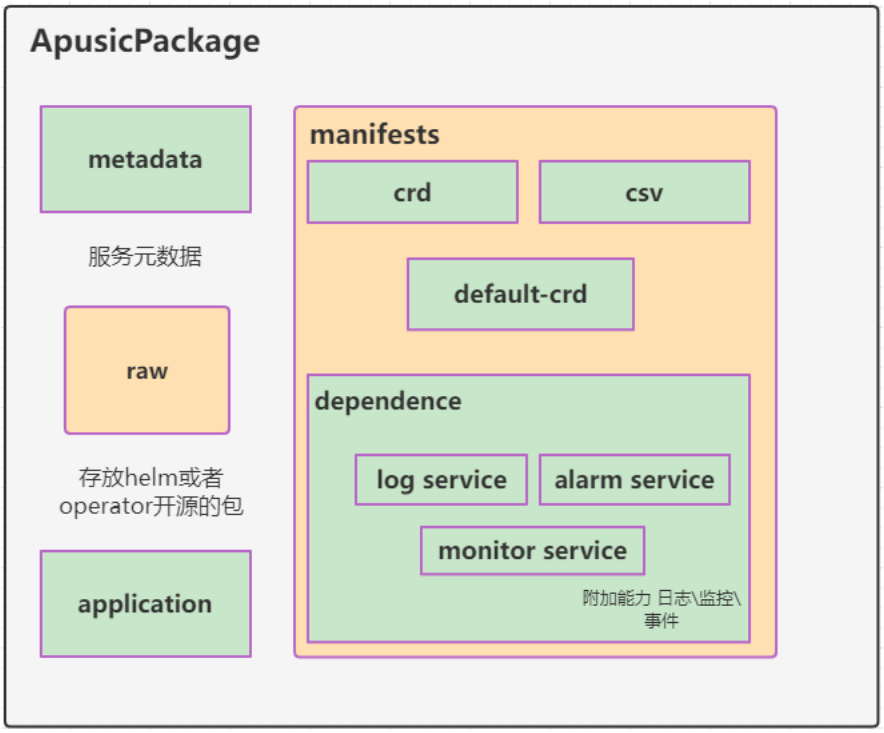
7.1 概述

服务开发者指南面向云原生服务的开发者，帮助开发者快速的将云原生服务转换为平台可使用的服务。平台通过服务包的形式，那么纳入云原生服务能力，并通过服务包定义，为云原生服务扩展可视化配置、一键部署、统一监控等能力。

7.2 服务规范

7.2.1 服务包定义

云原生服务中心（Operator Service Center，OSC）服务规范旨在给出一种与云平台解耦的云原生服务的标准定义，可描述云原生服务在分布式云的部署和治理。 使开发者能够快速为服务赋予部署和治理能力，实现一键部署和管理。



- 应用定义application：声明application服务包的安装、升级等有关服务包的相关信息。
- 服务元数据metadata：声明服务的核心数据，包括服务类型、支持架构、服务分类、使用场景等必要信息。
- 资源集合manifests：Resources资源定义的集合，对资源增强的服务能力配置，以及平台公共能力配置信息。
- 第三方兼容Raw：兼容常用第三方服务管理标准，主要是存放helm包及operator资源定义包。

7.2.2 服务包目录结构

```
apusic-package/
├─ metadata.yaml
├─ manifests/
|    ├─ crd.yaml
|    ├─ csv.yaml
|    ├─ dependence/
|    |    ├─ log-service.yaml
|    |    ├─ monitor-service.yaml
|    |    ├─ alarm-service.yaml
├─ raw/
└─ application.yaml
```

7.2.3 服务包规范

7.2.3.1 application

application.yaml 存放在{ACMP-PACKAGE}/application.yaml

- 示例

```yaml
apiVersion: operators.apusic.com/v1alpha1
kind: Application
metadata:
  name: redis-operator
spec:
  # 支持的类型 operator 或者 helm 的形式安装
  type: operator
  # 原生包中raw 文件里面的格式是什么格式 kubernetes helm kustomize 类型
  templateType: helm
  manifests:
    customResourceDefinitionName: crd.yaml
    clusterServiceVersionName: csv.yaml
    defaultCustomResource: default-cr.yaml
    capabilityRequirements:
      - apiVersion: operators.apusic.com/v1alpha1
        kind: LogConfig
        defaultPath: log_config.yaml
      - apiVersion: operators.apusic.com/v1alpha1
        kind: MonitorConfig
        defaultPath: monitor_config.yaml
```

- ApplicationSpec

参数名称	参数描述	类型	参数示例	默认值	必选
Upgrade	描述服务升级相关的内容	Upgrade			否
Manifests	描述Manifests相关内容	Manifests			是
DefaultConfiguration	默认配置	string			否
RawPath	定义的软件包的相对地址	string			否
Type	包类型	helm operator	helm		是
TemplateType	原生包中raw 文件里面的格式是什么格式 kubernetes helm kustomize 类型	helm kustomize kubernetes	helm		是

- Upgrade

参数名称	参数描述	类型	参数示例	默认值	必选
Replaces	升级基线版本	string			否
Skips	可以跳过升级的版本列表	[]string			否
SkipRange	可以跳过升级的版本，以正则表达式描述	string			否

Manifests

参数名称	参数描述	类型	参数示例	默认值	必选
CustomResourceDefinitionName	crd 的文件名称	string		crd.yaml	否
ClusterServiceVersionName	csv的文件名称	string		csv.yaml	否
DefaultCustomResource	default cr 的文件名称	string		default-cr.yaml	否

CapabilityRequirements	能力 需要在csd文件中声明capabilityRequirements	CapabilityRequirement			否

- CapabilityRequirement

参数名称	参数描述	类型	参数示例	默认值	必选
Name	名称	string			是
ApiVersion	Api版本	string			是
Kind	Kind	string			是
DefaultPath	yaml 的文件路径	string			是

7.2.3.2 metadata

元数据文件：{ACMP-Package}/metadata.yaml，配置的是服务包的基本信息，包括名称、版本、描述等相关信息。

- metadata参数

参数名称	参数描述	参数示例	必选
name	名称，符合正则表达式 ^[a-z][a-z0-9_-]*[a-z0-9]$	example-operator	是
version	版本 当前中间件或应用 的 operator的版本	1.0.0	是
appVersion	应用版本信息，长度小于64	2.1.2	否
alias	别名	example operator	否
displayName	对外显示的名字	Example operator	否
description	markdown格式的自定义描述	参考样例	否
detail	对外显示的名字	Example operator	否
source	包来源，取值范围：{ISV, OpenSource, apusicProvided}，分别表示独立服务供应商、开源服务、华为自有服务，仅支持配置一个值。默认值为OpenSource	fOpenSource	否
type	包类型，取值范围：{helm, operator}，仅支持配置一个值，默认值为operator	operator	否
containerImage	容器镜像	参见样例	否
repository	镜像仓库	docker.io	否
architecture	支持架构列表，取值范围：{x86_64, aarch64}，默认值为x86_64	参见样例	否
capabilities	能力(Runtime、Message、IntegratedDelivery、Database、Log、Monitor、BigData、DeveloperTool、Network、Security、AI)	参见样例	否
categories	分类列表，范围不限，如"AI, Database"	参见样例	否
devices	实例运行需要使用的硬件设备列表，取值范围：{CPU, GPU, NPU}，默认值为CPU	参见样例	否
industries	行业列表，范围不限，如"education,media"	参见样例	否
logo.mediatype	图标文件类型	参见样例	否
logo.path	图标图标文件路径	参见样例	否
maintainers	维护人员列表，包含name和email两个字段	参见样例	否
provider	提供者，包含name和url两个字段	参见样例	否
scenes	使用场景列表，取值列表：{Cloud, Edge}，分别表示公有云和边缘云场景，大小写敏感，默认值为Cloud	参见样例	否
links	附加说明链接列表，包含name和url两个字段，例如项目介绍链接	参见样例	否

- 元数据样例:

```yaml
name: example-operator
version: 1.0.0
appVersion: 2.1.2
alias: example-operator.v1.0.0
displayName: Example operator
description: example operator with an example instance and action
detail: ReadME.MD
source: OpenSource
type: operator
containerImage: demo/example-operator:1.0.0
repository: docker.io
architecture:
  - x86_64
  - aarch64
capabilities:
  - Monitor
categories:
  - Database
devices:
  - CPU
  - GPU
  - NPU
industries:
  - education
  - media
logo:
  base64data: iVBORw.....
  mediatype: image/png
maintainers:
  - email: test@test.com
    name: test
provider:
  name: Example provider
  url: https://example.com/
scenes:
  - Cloud
  - Edge
links:
  - name: example link
    url: http://github.com/3scale/prome
```

- detail文件规范
  datail文件为markdown格式，需按照章节规范对该文件进行编写，可在标准章节之后，添加自定义章节。

中间件名称

首行以#为标题，展示该中间件的名称。

中间件简介

需包含背景、概述、功能、受众章节，使用户能够快速对该中间件进行了解。

中间件参数说明

以表格方式提供，需要包含如下内容，可根据需要添加，但不可以裁剪说明列。

列名	描述	备注
名称	配置参数的名称或路径	

参数描述	参数的作用、以及参数配置项的具体解释	
必选	该参数是否必填	
默认值	该参数不填写时的取值	
参数范围	枚举类型使用/分割,数值类型需提供取值的上下限	

中间件组件说明

以表格方式提供，对中间件构成和依赖进行说明。必要时可以链接架构图。

列名	描述	备注
名称	中间件组件名称	
说明	中间件组件作用	

其它自定义章节

其它自定义内容，用户可根据需要自行增加章节，补充中间件描述信息。

detail文件样例

```
# 金蝶Apusic分布式缓存

## 中间件简介
金蝶Apusic分布式缓存软件（Apusic In-Memory Data Cache，简称：AMDC）是一款自主研发、高性能、高可用、可扩展的分布式缓存系统，提供数据缓存、缓存管控等能力，主要适用于高并发、分布式、高频数据存取等系统场景，为关键应用提供安全可靠的缓存支撑能力，并兼容Redis协议与持久化数据文件，能实现简单快捷平稳替换Redis

## 中间件参数说明

| 参数名                                    | 参数描述      |必选 | 默认值 | 参数范围   |
|-------------------------------------------|------------|---| ----  |----   |
| values.cluster.clusterReplicaNoFailover|自动故障恢复 'yes:此集群从节点不会参与自动故障转移过程，但是可以手动强制执行故障转移  no:此集群从节点参与自动故障转移过程'  |是  | yes  | yes/no|

## 中间件组件说明
* 组件1：ADMC运行节点，核心服务节点
* 组件2：AMDC哨兵，代理模式下应用通过哨兵节点进行流量分流

架构图[!http://www.apusic.com/image/amdc.png]

## 其它自定义章节
```

7.2.3.3 csv

在不同业务环境下，服务对平台有着不同的诉求，而大部分云原生服务为了保持通用性，一般仅提供一些核心能力，导致服务配置、服务监控、服务访问等能力和易用性有所缺失。

因为通过扩展CRD机制，通过声明配置的方式，在apusic-package包中体现。以增加云原生服务的适应能力，使其更加适合于业务。

CSV本身属于CustomServiceDefinition类型的资源，其符合k8s资源的通用定义方法。其apiVersion和kind是固定的，metadata.name可自行配置：

```
apiVersion: operators.apusic.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: helm-release
```

CSD的spec包含下面的配置：

配置项	配置描述	必选

spec.customresourcedefinitions	云原生使用的CRD信息，可以使用平台内置的或第三方的扩展	是
spec.owned.specDescriptors	界面配置的扩展描述	是
spec.owned.statusDescriptors	状态信息的扩展描述	是
spec.installModes	对于命名空间的限制配置	是

- customresourcedefinitions定义

支持平台内置对于Helm部署进行支持的CRD

```
  customresourcedefinitions:
    owned:
    - description: represents a Redis failover
      kind: HelmRelease
      name: helmreleases.operators.apusic.com
      version: v1alpha1
      resources:
      - kind: HelmRelease
        name: helmreleases.operators.apusic.com
        version: v1alpha1
```

配置项	配置描述	必选
owned.name	CRD名称，用于界面展示。	是
owned.kind	CRD类型，用于界面展示。	是
owned.description	CRD描述，用于界面展示。	是
owned.resources.name	CRD名称	是
owned.resources.kind	CRD类型	是
owned.resources.description	CRD描述	是

- 界面配置 支持通过配置方式，生成部署参数录入界面。组件包含多种配置，具体请参考表单控件章节

```
specDescriptors:
  - description: 'yes:此集群从节点不会参与自动故障转移过程，但是可以手动强制执行故障转移   no：此集群从节点参与自动故障转移过程'
    displayName: 自动故障恢复
    path: values.cluster.clusterReplicaNoFailover
    x-descriptors:
      - urn:alm:descriptor:com.tectonic.ui:select:yes
      - urn:alm:descriptor:com.tectonic.ui:select:no
```

配置项	配置描述	必选
displayName	控件显示名称	是
description	控件显示描述	是
path	控件对应的字段，控件会使用路径对应的值来展示或修改	是
x-descriptors	控件类型，具体参考"表单控件"章节	是

- 状态展示配置 支持通过配置的方式，生成界面参数展示界面。通过path从资源路径中获取资源值，并授予该组件，从而完成动态状态展示能力。

```
statusDescriptors:
  - description: broker-cs
    displayName: 数据端口
    path: services[1].spec.ports[0].port
    x-descriptors:
      - urn:alm:descriptor:text
```

配置项	配置描述	必选
displayName	控件显示名称	是
description	控件显示描述	是
path	控件对应的字段，控件会使用路径对应的值来展示或修改	是
x-descriptors	控件类型，具体参考"表单控件"章节	是

7.2.3.4 log_config

服务日志可以通过log_config进行配置，通过配置进行接入服务日志的收集

- 配置样例

```
  apiVersion: operators.apusic.com/v1alpha1
  kind: LogConfig
  metadata:
  annotations:
  controller-gen.kubebuilder.io/version: (devel)
  creationTimestamp: null
  name: amdc-operator-log-config
  spec:
  logConfigItems:
    - labelSelector:
      app.kubernetes.io/name: 3.2.4
      path: /var/log/amdc.log
```

配置项	配置描述	必选
name	日志配置名称	是
path	日志路径	是

7.2.3.5 monitor_config

- 配置样例 当服务提供提供了监控接口时，可以通过monitor配置实现对监控指标的采集和展示

```
apiVersion: operators.apusic.com/v1alpha1
kind: MonitorConfig
metadata:
  name: amdc-helm-cluster-config
  namespace: default
spec:
  monitorConfigItems:
    - labelSelector:
        app.kubernetes.io/name: amdc
      prometheus:
        port: "metrics"
        displayOptions:
          levels:
            - name: "amdc_memory_used_peak_bytes"
```

```
            groupLabel: pod
            selectedMetrics:
              - "amdc_memory_used_peak_bytes"
            metrics:
              - title: "已使用内存最大值"
                expr: 'amdc_memory_used_peak_bytes{namespace="{{.Namespace}}"}'
                unitType: "byte"
```

配置项	配置描述	必选
port	暴露的服务名	是
name	监控项目名称	是
groupLabel	分组标识	是
metrics.title	图标显示名称	是
metrics.expr	取值表达式	是
metrics.unitType	表达式类型	是

7.2.4 表单控件

7.2.4.1 specDescriptors

specDescriptors提供服务实例部署时的可视化配置功能

- 数值控件

```
- description: redis主节点副本数，不能小于3
  displayName: redis主节点副本数
  path: values.master.replicaCount
  x-descriptors:
    - urn:alm:descriptor:com.tectonic.ui:number
```

- 值选择控件 通过下拉列表为用户提供固定值的选择列表。

```
specDescriptors:
  - description: 'yes:此集群从节点不会参与自动故障转移过程，但是可以手动强制执行故障转移  no: 此集群从节点参与自动故障转移过程'
    displayName: 自动故障恢复
    path: values.cluster.clusterReplicaNoFailover
    x-descriptors:
      - urn:alm:descriptor:com.tectonic.ui:select:yes
      - urn:alm:descriptor:com.tectonic.ui:select:no
```

- 布尔控件

```
适用于布尔类型的值设置场景。
- description: 'true:启用管控台  false: 不启用管控台'
  displayName: 启用管控台
  path: values.dashboard.enabled
```

```
  x-descriptors:
    - urn:alm:descriptor:com.tectonic.ui:booleanSwitch
```

7.2.4.2 statusDescriptors

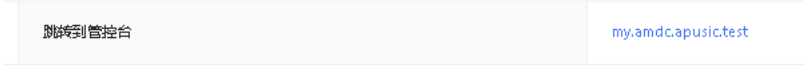
statusDescriptors提供服务实例部署后，状态字段动态展示的配置功能。

- 字符串控件

```
- description: broker-cs
  displayName: 数据端口
  path: services[1].spec.ports[0].port
  x-descriptors:
    - urn:alm:descriptor:text
```


- 跳转链接控件

```
- description: console
  displayName: 跳转到管控台
  path: ingress[0].spec.rules[0].host
  x-descriptors:
  - urn:alm:descriptor:org.w3:link
```



7.2.5 开源服务包规范

7.2.5.1 Helm规范

Helm的包格式被称为chart，描述Kubernetes相关资源的文件集合。chart包含：

- Chart.yaml
- values.yaml
- values.schema.json
- charts目录
- templates目录

```
{Helm}/
├─ Chart.yaml          #【必选】包含了chart信息文件，包括api版本、名称、描述、图标、类型、注释
├─ values.yaml         #【必选】参数值文档，为templates下的文档配置参数值
├─ values.schema.json  #【可选】一个使用JSON结构的values.yaml文件
├─ charts/             #【可选】包含chart依赖的其他chart
└─ templates           #【必选】模板目录，当和values 结合时，可生成有效的Kubernetes manifest文件
    └─ xxx.yaml        #【可选】Helm包的Kubernetes manifest若干文件
```

更多介绍参见官网Helm。

7.2.5.2 Operator规范

Operator包含三类核心文件：

- 软件包清单：package.yaml
- 集群服务版本：csv.yaml
- 自定义资源：crd.yaml

```
{Operator-Package}
├─ xxx.package.yaml            #【必选】软件包清单
└─ {version}                   #【必选】version目录，例如21.7.1
    ├─ xxx_csv.yaml            #【必选】ClusterServiceVersion集群服务版本文件，可多个。
    └─ xxx_crd.yaml            #【必选】CustomResourceDefinition自定义资源定义文件，可多个。
```

更多介绍参见官网Operator Framework。

7.3 服务接入

7.3.1 服务接入要求

7.3.1.1 部署要求

- 软件形态
  纳入的中间件应转换为Helm或Operation形态，提供统一的config.yaml，以便实现平台实现统一配置。
  软件镜像应统一上传到平台规定的镜像仓库中。

- 架构支持
  纳入的中间件应至少支持AMD64或ARM64中的一种。若软件同时支持多种架构时，应在软件描述中明确支持的架构种类，便于用户检索。
  镜像建议采用多架构融合镜像，减少配置部署时，涉及到架构配置的内容。

- 版本要求
  中间件应具有明确的发布版本，提供的镜像必须具有明确的版本，不允许使用latest版本。以免因版本不一致造成问题。

7.3.1.2 测试要求

7.3.1.2.1 功能测试报告

功能测试报告应涵盖如下内容：

1. 测试概述
   本报告是针对软件产品XXX的功能测试报告，目的是总结测试过程中发现的问题，评估软件产品的功能质量，提供改进建议。
   测试范围包括软件产品的主要功能模块。
   测试方法采用黑盒测试，根据需求文档和设计文档编写测试用例，使用自动化测试工具或手工测试工具执行测试用例，记录测试结果和缺陷。
   测试环境包括测试服务器、测试数据库、测试网络、测试浏览器等。
   测试时间为XXXX年XX月XX日至XXXX年XX月XX日。
2. 测试结果
   测试用例总数为XXX个，执行用例数为XXX个，通过用例数为XXX个，失败用例数为XXX个，未执行用例数为XXX个。
   缺陷总数为XXX个，严重缺陷数为XXX个，一般缺陷数为XXX个，轻微缺陷数为XXX个。
   功能测试覆盖率为XX.XX%，功能测试通过率为XX.XX%，功能缺陷密度为XX.XX%。
   主要测试问题：
   1.测试问题描述
   2.测试问题描述
3. 测试分析
   软件产品的功能基本满足需求，但仍存在一些缺陷，需要进一步修改和优化。
   缺陷主要集中在XX模块和XX模块，可能与XXX有关，需要开发人员进行排查和修复。
   功能测试覆盖率和通过率达到了预期目标，但缺陷密度较高，说明软件产品的功能质量有待提高。
   建议在下一轮测试中增加未执行用例的执行，重点关注严重缺陷和一般缺陷的修复情况，同时进行回归测试和兼容性测试。

7.3.1.2.2 高可用测试报告

1. 测试概述
   本报告是针对软件产品XXX的高可用测试报告，目的是评估软件产品在异常情况下的可用性和稳定性，验证软件产品的容错能力和恢复能力。
   测试范围包括软件产品的关键业务流程。
   测试方法采用故障注入法，模拟各种异常情况，如服务器宕机、网络断开、数据库故障、并发压力等，观察软件产品的表现和响应。
   测试环境包括测试服务器集群、负载均衡器、数据库集群、网络设备等。
   测试时间为XXXX年XX月XX日至XXXX年XX月XX日。
2. 测试结果
   测试场景总数为XXX个，执行场景数为XXX个，通过场景数为XXX个，失败场景数为XXX个。

软件产品在以下异常情况下仍能正常运行：
单台服务器宕机；
单台数据库故障；
网络延迟；

3. 测试分析
软件产品的性能满足需求，但仍存在一些缺陷，需要进一步修改和优化。
缺陷主要集中在XX模块和XX模块，可能与XXX有关，需要开发人员进行排查和修复。

7.3.1.2.3 性能测试报告

1. 测试概述
本报告为XXX软件产品的性能测试报告，目的在于评估软件产品的性能指标，如响应时间、吞吐量、资源利用率等，以及在不同负载条件下的系统表现。
测试范围包括软件产品的关键业务流程。
测试方法采用采用负载测试、压力测试、稳定性测试等性能测试方案，模拟不同的用户场景和并发量，观察系统的性能变化。
测试环境包括测试服务器集群、负载均衡器、数据库集群、网络设备等。
测试时间为XXXX年XX月XX日至XXXX年XX月XX日。
2. 测试结果
测试场景总数为XXX个，执行场景数为XXX个
TC01：模拟100个用户同时登录系统，持续10分钟
TC02：模拟200个用户同时进行查询操作，持续20分钟
TC03：模拟300个用户同时进行增删改操作，持续30分钟
3. 测试分析
根据测试结果，对软件产品的性能进行如下评价：
软件产品的平均响应时间在1秒以内，符合用户的期望和要求。
软件产品的最大响应时间在3秒以内，没有出现超时或卡顿的现象。
软件产品的吞吐量随着用户数量的增加而增加，表明系统具有良好的扩展性。
软件产品的CPU利用率和内存利用率在合理范围内，没有出现过高或过低的情况。
综上所述，软件产品的性能达到了预期目标，可以满足用户的需求和场景。

7.3.1.2.4 安全测试报告

1. 测试概述
本报告为软件产品的安全测试报告，目的在考察软件产品的安全性、测试结论以及测试建议。 测试方法采用账号安全管理、权限管理、安全日志、访问控制安全、输入安全、缓冲区溢出、SQL注入、跨站脚本攻击等安全测试方案，观察系统的安全性。
测试环境包括测试服务器集群、负载均衡器、数据库集群、网络设备等。
测试时间为XXXX年XX月XX日至XXXX年XX月XX日。

2. 测试结果
共发现安全问题XXX个，其中高级别漏洞XXX个

缺陷编号	缺陷描述	缺陷等级	缺陷状态	影响模块	影响功能
1	账号密码强度过低，容易被破解	高	新建	账号管理	注册登录
2	验证码识别率低，无法有效防止暴力破解	中	新建	账号管理	注册登录
3	普通用户可以访问管理员角色的页面，造成权限泄露	高	新建	权限管理	角色切换

3. 测试分析
根据本次测试结果及缺陷分析，我们得出以下测试结论： 软件产品不存在高等级以上的软件漏洞，符合产品使用条件。

7.3.1.3 安全要求

为保证软件服务安全，接入服务需满足如下安全要求：

访问安全

- 不允许设置非空密码，若支持匿名访问的系统，需在服务说明中进行说明，并标识风险
- 密码设置应设置平台密码库，并支持密码库修改时，同步修改服务内的密码。
- 直接发布到外部的服务应支持https，否则应通过平台服务网关进行统一发布。
- 软件发布时，应通过安全扫描，并提供安全测试报告。
- 涉及到用户管理的，应尽量使用平台提供的用户管理体系进行统一认证管理

存储安全

- 除临时数据外，服务应统一使用平台提供公共存储，便于统一进行备份、管理和恢复。

- 涉及到用户敏感信息的，需要加密存储。
- 需支持存储版本号，不允许直接覆盖历史存储文件，导致备份记录丢失

7.3.1.4 操作要求

中间件需以标准方式提供中间件的标准化操作能力，以便平台进行统一接入与管理

部署、卸载与升级、回滚

部署、卸载与升级、回滚基于helm或Operator的标准操作实现。

启动、停止

如中间件提供启动停止功能，则需要提供op-产品名-startup.yaml以及op-产品名-shutdown.yaml配置。
配置支持Pod、conjob或自定义类型。
平台提供启动、停止的调用接口，调度相关配置行为，但不保证执行成功。

备份、还原

如中间件提供启动备份还原，则需要提供op-产品名-backup.yaml以及op-产品名-recovery.yaml配置。
配置支持Pod、conjob或自定义类型。
平台提供备份、还原的调用接口，调度相关配置行为，但不保证执行成功。

其它

如中间件提供其它自定义行为，则按照op-产品名-操作行为.yaml进行提供。 平台提供相关的调用接口，调度相关配置行为，但不保证执行成功。

7.3.2 服务包开发管理工具

olmctl是平台为了开发和管理服务包而提供的工具软件。提供服务包生成、服务上传、仓库管理等功能。
工具能够将Helm或者Operator包转为标准服务包，并发布到云平台中。

介绍 获取olmclt工具包，在windows下执行olmctl，在linux下执行./olmctl即可查看工具使用指导。

```
The OLM(operator lifecycle manager) package manager for Kubernetes.

Usage:
olmctl [command]

Available Commands:
completion  Generate the autocompletion script for the specified shell
help        Help about any command
init        init acmp package
oc          operatorcomponent 中间件实例生命周期
package     服务包管理
repo        仓库管理
version     Operator Lifecycle Manager Version
```

创建服务包 执行olmctl creat kafka-operation,即可生成一个名称为kafka-operation的标准服务包。
目录结构如下：

```
kafka-operation
├─manifests
│  └─dependence
└─raw
  └─kafka-operati
  ├─charts
  └─templates
    └─tests
```

服务包内的具体文件格式，参考服务规范章节。

7.3.3 Helm接入

将原始helm放入raw目录中，并根据服务规范进行服务包配置。

7.3.4 Operator接入

将原始Operator放入raw目录中，并根据服务规范进行服务包配置。

7.4 扩展

7.4.1 开发Operator

7.4.1.1 先决条件

- 已安装Kubebuilder
- 已安装go v1.16+
- 已安装make
- 已安装docker 17.03+
- 已安装Kubernetes v1.18.0+

7.4.1.2 从应用文件制作镜像

下载依赖包

- 下载JRE：https://download.java.net/openjdk/jdk11/ri/openjdk-11+28_linux-x64_bin.tar.gz。
- 下载Apache Kafka二进制包：https://archive.apache.org/dist/kafka/2.7.0/kafka_2.13-2.7.0.tgz。
- 下载Kafka监控指标上报库：https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.15.0/jmx_prometheus_javaagent-0.15.0.jar。

创建配置文件

1. 创建初始化ZooKeeper的配置启动参数脚本zkGenConfig.sh。

```
#!/bin/bash

ZK_CONF_DIR=${ZK_CONF_DIR:-"/opt/kafka_2.13-2.7.0/config"}
ZK_CONFIG_FILE="$ZK_CONF_DIR/zookeeper.properties"
ZK_DATA_DIR=${ZK_DATA_DIR:-"/var/lib/zookeeper/zk/${NAMESPACE}/${INSTANCE}/${POD_NAME}/data"}
ID_FILE="$ZK_DATA_DIR/myid"
ZK_REPLICAS=${ZK_REPLICAS:-"3"}
ZK_CLIENT_PORT=${ZK_CLIENT_PORT:-2181}
ZK_SERVER_PORT=${ZK_SERVER_PORT:-2888}
ZK_ELECTION_PORT=${ZK_ELECTION_PORT:-3888}

# 1. 生成 myid
echo "Generate myid"

HOST=$(hostname -s)
DOMAIN=$(hostname -d)

if [[ $HOST =~ (.*)-([0-9]+)$ ]]; then
  NAME=${BASH_REMATCH[1]}
  ORD=${BASH_REMATCH[2]}
else
  echo "Failed to extract ordinal from hostname $HOST"
  exit 1
fi

MY_ID=$((ORD + 1))
echo "MY_ID=$MY_ID"

mkdir -p $ZK_DATA_DIR
rm -f $ID_FILE
if [ ! -f $ID_FILE ]; then
  echo $MY_ID >>$ID_FILE
```

```
fi

# 2. 副本数量
if [ -z $ZK_REPLICAS ]; then
  echo "The environment variable ZK_REPLICAS is not exist."
  exit 1
fi

echo "ZK_REPLICAS=$ZK_REPLICAS"

# 3. 创建配置文件
rm -f $ZK_CONFIG_FILE
echo "tickTime=2000" >>$ZK_CONFIG_FILE
echo "initLimit=5" >>$ZK_CONFIG_FILE
echo "syncLimit=2" >>$ZK_CONFIG_FILE
echo "dataDir=$ZK_DATA_DIR" >>$ZK_CONFIG_FILE
echo "clientPort=$ZK_CLIENT_PORT" >>$ZK_CONFIG_FILE
for ((i = 1; i <= $ZK_REPLICAS; i++)); do
  echo "server.$i=$NAME-$((i - 1)).$DOMAIN:$ZK_SERVER_PORT:$ZK_ELECTION_PORT" >> $ZK_CONFIG_FILE
done
```

2. 创建监控指标上报配置文件metrics-config.yaml

```
startDelaySeconds: 0
lowercaseOutputName: false
lowercaseOutputLabelNames: false
rules:
  - pattern: kafka.controller<type=KafkaController, name=GlobalTopicCount><>Value
    name: kafka_controller_KafkaController_Value
    type: GAUGE
    labels:
      name: GlobalTopicCount
  - pattern: kafka.controller<type=KafkaController, name=GlobalPartitionCount><>Value
    name: kafka_controller_KafkaController_Value
    type: GAUGE
    labels:
      name: GlobalPartitionCount
  - pattern: kafka.server<type=ReplicaManager, name=LeaderCount><>Value
    name: kafka_server_ReplicaManager_Value
    type: GAUGE
    labels:
      name: LeaderCount
  - pattern: kafka.server<type=ReplicaManager, name=PartitionCount><>Value
    name: kafka_server_ReplicaManager_Value
    type: GAUGE
    labels:
      name: PartitionCount
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesInPerSec><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: BytesInPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesOutPerSec><>Count
    name: kafka_server_BrokerTopicMetrics_Count
```

```
    type: GAUGE
    labels:
      name: BytesOutPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=MessagesInPerSec><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: MessagesInPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalProduceRequestsPerSec><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: TotalProduceRequestsPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalFetchRequestsPerSec><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: TotalFetchRequestsPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesInPerSec><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: BytesInPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesOutPerSec><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: BytesOutPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=MessagesInPerSec><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: MessagesInPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalProduceRequestsPerSec><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: TotalProduceRequestsPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalFetchRequestsPerSec><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: TotalFetchRequestsPerSec
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesInPerSec, topic=(.+)><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: BytesInPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesOutPerSec, topic=(.+)><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: BytesOutPerSec
```

```
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=MessagesInPerSec, topic=(.+)><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: MessagesInPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalProduceRequestsPerSec, topic=(.+)><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: TotalProduceRequestsPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalFetchRequestsPerSec, topic=(.+)><>Count
    name: kafka_server_BrokerTopicMetrics_Count
    type: GAUGE
    labels:
      name: TotalFetchRequestsPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesInPerSec, topic=(.+)><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: BytesInPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=BytesOutPerSec, topic=(.+)><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: BytesOutPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=MessagesInPerSec, topic=(.+)><>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: MessagesInPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalProduceRequestsPerSec, topic=(.+)>
<>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: TotalProduceRequestsPerSec
      topic: "$1"
  - pattern: kafka.server<type=BrokerTopicMetrics, name=TotalFetchRequestsPerSec, topic=(.+)>
<>OneMinuteRate
    name: kafka_server_BrokerTopicMetrics_OneMinuteRate
    type: GAUGE
    labels:
      name: TotalFetchRequestsPerSec
      topic: "$1"
  - pattern: kafka.log<type=Log, name=(.+), topic=(.+), partition=(.+)><>Value
    name: kafka_log_Log_Value
    type: GAUGE
```

```
    labels:
      name: "$1"
      topic: "$2"
      partition: "$3"
  - pattern: java.lang<type=OperatingSystem><>SystemCpuLoad
    name: os_SystemCpuLoad
    type: GAUGE
    valueFactor: 100
  - pattern: java.lang<type=OperatingSystem><>ProcessCpuLoad
    name: os_ProcessCpuLoad
    type: GAUGE
    valueFactor: 100
  - pattern: java.lang<type=OperatingSystem><>TotalPhysicalMemorySize
    name: os_TotalPhysicalMemorySize
    type: GAUGE
  - pattern: java.lang<type=OperatingSystem><>FreePhysicalMemorySize
    name: os_FreePhysicalMemorySize
    type: GAUGE
```

3.创建Dockerfile

```
FROM centos:latest

ENV PATH="/opt/jre-11.0.11/bin:${PATH}" \
    KAFKA_HOME="/opt/kafka_2.13-2.7.0"

ADD openjdk-11+28_linux-x64_bin.tar.gz /opt/
ADD kafka_2.13-2.7.0.tgz /opt/
ADD zkGenConfig.sh ${KAFKA_HOME}/bin
ADD jmx_prometheus_javaagent-0.15.0.jar ${KAFKA_HOME}/libs
ADD metrics-config.yaml ${KAFKA_HOME}/config

RUN ln -s ${KAFKA_HOME} /opt/kafka \
    && chmod u+x ${KAFKA_HOME}/bin/zkGenConfig.sh \
    && sed -i "/kafka-run-class.sh/i\\export JMX_PORT=9999" ${KAFKA_HOME}/bin/kafka-server-start.sh \
    && sed -i "/kafka-run-class.sh/i\\export KAFKA_OPTS=\"\${KAFKA_OPTS} -
javaagent:\$base_dir\/\.\.\/libs/jmx_prometheus_javaagent-
0.15.0.jar=9404:\$base_dir\/\.\.\/config\/metrics-config.yaml\"" ${KAFKA_HOME}/bin/kafka-server-start.sh

WORKDIR $KAFKA_HOME
```

制作容器镜像 构建容器镜像命令如下：

```
$ docker build -t kafka:v2.7.0 .
```

7.4.1.3 安装Kubebuilder

Kubebuilder是基于custom resource definitions(CRDs)构建Kubernetes APIs的框架，有如下两种安装方法。

- 安装方法一：根据kubebuilder官网提供的go语言版本和节点系统架构下载。

```
# go env GOOS -- 获取操作系统类型，例如：linux等
# go env GOARCH -- 获取系统架构，例如：arm或amd64等
$ curl -L -o kubebuilder https://go.kubebuilder.io/dl/latest/$(go env GOOS)/$(go env GOARCH)
$ chmod +x kubebuilder && mv kubebuilder /usr/local/bin/
```

- 安装方法二：避免因证书导致无法通过curl进行下载。

```
# 通过git clone及本地编译的方法安装kubebuilder可以避免因操作系统类型、系统架构输入错误导致
# kubebuilder二进制文件因操作系统和系统架构导致错误导致的无法使用。同时，通过git操作避免
# 在使用curl过程中需要对开发环境进行证书配置等操作
$ git clone https://github.com/kubernetes-sigs/kubebuilder.git
Cloning into 'kubebuilder'...
remote: Enumerating objects: 49241, done.
remote: Counting objects: 100% (2995/2995), done.
remote: Compressing objects: 100% (1087/1087), done.
remote: Total 49241 (delta 1897), reused 2679 (delta 1741), pack-reused 46246
Receiving objects: 100% (49241/49241), 63.87 MiB | 1.84 MiB/s, done.
Resolving deltas: 100% (25723/25723), done.
Updating files: 100% (1230/1230), done.

$ cd kubebuilder
$ git tag # 列出git目录下所有tag
release-0.1.6
tools-1.10.1
tools-1.11.0
v0.1.10
v0.1.11
v0.1.12
....
$ git checkout v3.2.0 # 目前最新tag为v3.2.0，可根据开发需求选择合适的版本
Note: switching to 'v3.2.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at b7a730c8 Merge pull request #2409 from camilamacedo86/release-3

# 检查是否切换到想要的版本
$ git branch
* (HEAD detached at v3.2.0)
  master

# 编译kubebuilder二进制文件
$ make build
go build -ldflags " -X main.kubeBuilderVersion=v3.2.0 -X main.goos=linux -X main.goarch=amd64 -X
main.gitCommit=b7a730c84495122a14a0faff95e9e9615fffbfc5 -X main.buildDate=2021-12-14T06:30:08Z " -o
bin/kubebuilder ./cmd
```

```
go: downloading github.com/gobuffalo/flect v0.2.3
go: downloading sigs.k8s.io/controller-tools v0.7.0
go: downloading sigs.k8s.io/kustomize/kyaml v0.10.21
go: downloading k8s.io/api v0.22.2
go: downloading k8s.io/apiextensions-apiserver v0.22.2

# 复制二进制文件到指定目录，方便后期使用
$ cp bin/kubebuilder /usr/local/bin/
```

7.4.1.4 构建Operator

CRD(CustomResourceDefinition)是一种自定义的Kubernetes资源，在定义CRD后，可通过在集群内创建对应 CR(CustomResource)对应用进行统一管理。格式示例请参见CRD典型格式。

7.4.1.4.1 CRD介绍

在定义CRD时，需同时定义基于OpenAPI的校验规则，其中包含创建CR时的字段及取值范围，以便校验用户创建的CR中字段值的合法性。使用Kubebuilder，可通过在API中定义Marker，自动生成spec.validation.openAPIV3Schema。

7.4.1.4.2 CRD字段说明

- group，一般为组织名称，如：osctest。
- API版本，第一个版本一般为v1。
- scope，Namespaced级别，修改为Cluster级别，则集群内仅可以创建一个同名的CR。仅Cluster级别的operator可管理同级别CRD。
- API名称，假设应用名称为kafka，则对应的多种名称格式如下：
- API名称：kind: kafka，创建API的时候使用。

7.4.1.4.3 构建CR

通过CustomResource(CR)的方式创建新资源类型，在CR中为应用定义参数和状态属性。

参数

- size：应用实例包含的实例数量。
- image：应用的容器镜像地址。
- storage：应用数据存储相关配置。

状态

- phase：应用实例安装状态。
- server：应用访问地址。 以创建一个Kafka应用，指定其Pod数量为3为例，设计CR如下：

7.4.1.4.4 创建OPERATOR项目

Operator是Kubernetes的扩展软件，通过定制资源管理应用和其他组件，实现一定自动运维能力。可以在不改动Kubernetes源码的情况下，通过一个或多个Operator来扩展集群能力，遵照自身业务需求、场景等灵活开发，节省运维成本。流程包括安装Kubernetes、构建Operator和实现Operator。

创建Operator 本章节以应用名称为kafka、取项目名称为kafka-operator为例，说明如何创建Operator。

```
# 在 GOPATH 下新建工程目录
$ mkdir -p $GOPATH/src/kafka-operator
$ cd $GOPATH/src/kafka-operator
# 使用kubebuilder初始化脚手架框架，并设定域名为apusic.com
$ kubebuilder init --domain apusic.com
Writing kustomize manifests for you to edit...
Writing scaffold for you to edit...
Get controller runtime:
# 检查如果环境所需要的依赖包不满足kubebuilder要求，kubebuilder会下载相关依赖包，开发者可根据实际情况在项目的go.mod中更改依赖包
go get sigs.k8s.io/controller-runtime@v0.10.0
go: downloading sigs.k8s.io/controller-runtime v0.10.0
go: downloading k8s.io/client-go v0.22.1
go: downloading k8s.io/utils v0.0.0-20210802155522-efc7438f0176
go: downloading k8s.io/component-base v0.22.1
go: downloading k8s.io/apiextensions-apiserver v0.22.1
Update dependencies:
```

```
go mod tidy
go: downloading github.com/benbjohnson/clock v1.1.0
Next: define a resource with:
kubebuilder create api
```

创建Operator脚手架工程，目录结构如下：

```
kafka-operator/
├── Dockerfile
├── Makefile
├── PROJECT
├── config
│   ├── default
│   │   └── ...
│   ├── manager
│   │   └── ...
│   ├── prometheus
│   │   └── monitor.yaml
│   └── rbac
│       └── ...
├── go.mod
├── go.sum
├── hack
│   └── boilerplate.go.txt
└── main.go
```

Operator作用域 Operator的作用域分namespace级和cluster级，cluster级的operator可以监听和管理任意namespace的资源。使用kubebuilder init命令默认初始化cluster级的operator，仅支持cluster级的operator管理同级别的CRD。监听多个namespace，main.go修改代码如下：

```
// namespace 列表
namespaces := []string{"foo", "bar"}
mgr, err := ctrl.NewManager(ctrl.GetConfigOrDie(), ctrl.Options{
    Scheme:             scheme,
    MetricsBindAddress: metricsAddr,
    Port:               9443,
    LeaderElection:     enableLeaderElection,
    LeaderElectionID:   "f1c5ece8.apusic.com",
    NewCache:           cache.MultiNamespacedCacheBuilder(namespaces),  // 指定多个 namespace
})
```

7.4.1.4.5 创建API和CONTROLLER

根据以上设计的CR，在项目工程中创建对应的CRD类型：

```
$ kubebuilder create api --group osctest --version v1 --kind kafka
Create Resource [y/n]
y
Create Controller [y/n]
y
Writing kustomize manifests for you to edit...
Writing scaffold for you to edit...
api/v1/kafka_types.go
controllers/kafka_controller.go
Update dependencies:
go mod tidy
Running make:
```

```
make generate
go: creating new go.mod: module tmp
Downloading sigs.k8s.io/controller-tools/cmd/controller-gen@v0.7.0
go get: added sigs.k8s.io/controller-tools v0.7.0
/mnt/d/repository/Go/GOPATH/src/kafka-operator/bin/controller-gen
object:headerFile="hack/boilerplate.go.txt" paths="./..."
Next: implement your new API and generate the manifests (e.g. CRDs,CRs) with:
make manifests
```

该命令主要执行以下动作：

1. 在PROJECT文件中增加API资源声明。

```
domain: apusic.com
layout:
- go.kubebuilder.io/v3
projectName: kafka-operator
repo: kafka-operator
resources:
- api:
    crdVersion: v1
    namespaced: true
  controller: true
  domain: apusic.com
  group: osctest
  kind: kafka
  path: kafka-operator/api/v1
  version: v1
version: "3"
```

2. 新增CRD及CR的描述文件。

```
kafka-operator
└── config
    ├── crd
    │   ├── kustomization.yaml
    │   ├── kustomizeconfig.yaml
    │   └── patches
    │       ├── cainjection_in_kafkas.yaml
    │       └── webhook_in_kafkas.yaml
    ├── rbac # 增加文件
    │   ├── kafka_editor_role.yaml
    │   └── kafka_viewer_role.yaml
    └── samples
        └── osctest_v1_kafka.yaml
```

3. 新增api目录，包含CRD的类型定义。

```
kafka-operator
└── api
    └── v1
        ├── groupversion_info.go
        ├── kafka_types.go
        └── zz_generated.deepcopy.go
```

4. 新增controllers目录，包含控制器的业务逻辑。

```
kafka-operator
└── controllers
    ├── kafka_controller.go
    └── suite_test.go
```

5. 在main.go的Manager启动时，创建Controller。

```
if err = (&controllers.kafkaReconciler{
    Client: mgr.GetClient(),
    Scheme: mgr.GetScheme(),
}).SetupWithManager(mgr); err != nil {
    setupLog.Error(err, "unable to create controller", "controller", "kafka")
    os.Exit(1)
}
```

7.4.1.5 实现Operator

7.4.1.5.1 定义API

在设计的CR spec中，包含size、image、storage属性，因此需要修改api/v1/Kafka_types.go中kafkaSpec和KafkaStatus部分，为应用定义参数和状态属性。

```
package v1

import (
corev1 "k8s.io/api/core/v1"
"k8s.io/apimachinery/pkg/api/resource"
metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// KafkaSpec defines the desired state of Kafka
type KafkaSpec struct {
// INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
// Important: Run "make" to regenerate code after modifying this file

    // +kubebuilder:validation:Minimum=1
    // +kubebuilder:validation:Maximum=3
    // +kubebuilder:validation:ExclusiveMaximum=false  // 包含最大值
    Size    int32        `json:"size"`    // Kafka 应用包含的 broker 数量
    Image   string       `json:"image"`   // Kafka 镜像地址
    Storage *StorageSpec `json:"storage"` // Kafka 数据存储相关配置
}

type StorageSpec struct {
Class       string                            `json:"class"`
AccessModes corev1.PersistentVolumeAccessMode `json:"accessModes"`
Size        resource.Quantity                 `json:"size"`
// 华为公有云 EVS 场景需额外指定 diskType, region, zone
DiskType string `json:"diskType,omitempty"`  // omitempty 表示可以为空
Region   string `json:"region,omitempty"`
Zone     string `json:"zone,omitempty"`
}

// KafkaStatus defines the observed state of Kafka
type KafkaStatus struct {
// INSERT ADDITIONAL STATUS FIELD - define observed state of cluster
```

```
// Important: Run "make" to regenerate code after modifying this file

    Phase  string `json:"phase,omitempty"`  // Kafka 实例安装状态
    Server string `json:"server,omitempty"` // Kafka 访问地址
}
```

使用Kubebuilder，可通过在API结构的属性上定义Markers，自动生成CRD中的spec.validation.openAPIV3Schema，即基于OpenAPI的校验规则，以便校验用户创建的CR中字段值的合法性。如上述设置size属性的最小值和最大值：

```
// +kubebuilder:validation:Minimum=1
// +kubebuilder:validation:Maximum=3
```

详细使用方法参见：CRD Validation。每次修改API定义后，需要执行命令自动重新生成代码和CRD：

```
$ make generate
$ make manifests
```

7.4.1.5.2 实现CONTROLLER

Controller**实现背景** 在创建Kafka实例时，Kafka-Operator需要创建的Kubernetes资源如下：

- 1个StatefulSet，包含3个Pod分别启动ZooKeeper；
- 1个Service，用来暴露ZooKeeper访问地址；
- 1个StatefulSet，包含3个Pod分别启动Kafka broker；
- 1个Service，用来暴露Kafka访问地址；
- 1个Deployment，包含1个Pod启动KafkaManager；
- 1个Service，用来暴露KafkaManager访问地址。 在创建API的时候，SDK已自动创建controllers/kafka_controller.go，为SetupWithManager添加对Service、StatefulSet、Deployment的监听和管理。

```
import (
    "context"
    "fmt"

    "github.com/go-logr/logr"
    appsv1 "k8s.io/api/apps/v1"
    corev1 "k8s.io/api/core/v1"
    "k8s.io/apimachinery/pkg/api/errors"
    "k8s.io/apimachinery/pkg/runtime"
    "k8s.io/apimachinery/pkg/types"
    ctrl "sigs.k8s.io/controller-runtime"
    "sigs.k8s.io/controller-runtime/pkg/client"

    osctestv1 "kafka-operator/api/v1"
)

func (r *KafkaReconciler) SetupWithManager(mgr ctrl.Manager) error {
    return ctrl.NewControllerManagedBy(mgr).
        For(&osctestv1.Kafka{}).
        Owns(&corev1.Service{}).
        Owns(&appsv1.StatefulSet{}).
        Owns(&appsv1.Deployment{}).
        Complete(r)
```

调度逻辑 对于Kubernetes集群内任何Kafka或Service、StatefulSet、Deployment的变化，Controller都会监听到，并生成事件，触发Reconcile()方法。需要在Reconcile()方法中实现协调逻辑，创建Service、StatefulSet等资源，并更新应用实例状态，实现的过程中，可以参考：

1. 查询资源：控制器使用controller-runtime库中的Client实现对Kubernetes资源的增查改删，示例代码参见：example_test.go。
2. 创建资源：使用Go语言调用Kubernetes API创建资源，可参考Kubernetes API Reference。

3. 设置关联：为创建的Kubernetes资源设置ownerReferences，以便其能在CR删除时被级联删除，可参考如下代码。

```
// 为指定命名空间和名称的 secret 设置 owner，返回 secret
func (k *K8sClient) SetSecretOwner(cr *oscv1.Kafka, secretName string) (*corev1.Secret, error) {
    ctx := context.Background()

    secret, err := k.GetSecretByNamespaceName(cr.Namespace, secretName)
    if err != nil {
        return nil, err
    }
    if secret.OwnerReferences == nil {
        patch := client.MergeFrom(secret.DeepCopy())
        secret.OwnerReferences = []metav1.OwnerReference{
            *metav1.NewControllerRef(cr, schema.GroupVersionKind{
                Group:   cr.GroupVersionKind().Group,
                Version: cr.GroupVersionKind().Version,
                Kind:    cr.Kind,
            }),
        }
        if err := k.Patch(ctx, secret, patch); err != nil {
            return nil, err
        }
    }
    return secret, nil
}
```

4. 如果一个动作的处理时间较长，为了避免Reconcile阻塞，需要使请求返回并重新排队，有四种方法：

```
// 请求成功，不再排队
return ctrl.Result{}, nil
// 请求失败，重新加入队列
return ctrl.Result{}, err
// 请求因某种原因需要重新加入队列
return ctrl.Result{Requeue: true}, nil
// 请求因某种原因，需要在指定时间后重新加入队列
return ctrl.Result{RequeueAfter: time.Second*5}, nil
```

5. 当删除CR时，如果需要预先清理Kubernetes之外的资源，此时仅凭ownerReferences无法实现，可以利用finalizers特性。

挂载存储 应用实例可以使用本地磁盘存储（HostPath），在Kubernetes中，通过PersistentVolume(PV)方式挂载存储。

RBAC权限管理 Operator使用基于角色的访问控制机制对Kubernetes中的资源进行访问，因此，要保证Operator正确的运行，需要使用管理员权限的帐号对Operator进行授权，即预先创建对应的role和rolebinding。

在Reconcile()上添加markers，增加Operator对于Service、StatefulSet、Deployment、Pod的管理权限，SDK即可自动生成对应的role和rolebinding等资源描述文件。

```
//
+kubebuilder:rbac:groups=osctest.huawei.com,resources=kafkas,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=osctest.huawei.com,resources=kafkas/status,verbs=get;update;patch
// +kubebuilder:rbac:groups=core,resources=services,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=apps,resources=statefulsets,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=core,resources=pods,verbs=get;list;watch
// +kubebuilder:rbac:groups=core,resources=persistentvolumeclaims,verbs=get;list;watch;update

func (r *KafkaReconciler) Reconcile(req ctrl.Request) (ctrl.Result, error) {
```

```
    // ...
}
```

如果Operator为namespace级别的，则需要还添加namespace，如：

```
//
+kubebuilder:rbac:groups=osctest.huawei.com,namespace=example,resources=kafkas,verbs=get;list;watch;create;u
//
+kubebuilder:rbac:groups=osctest.huawei.com,namespace=example,resources=kafkas/status,verbs=get;update;patch
```

7.4.1.5.3 生成代码和资源描述文件

修改api/v1/memcached_types.go或controller中的markers之后，需要重新生成代码和资源描述文件。

```
# 生成 api/v1/zz_generated.deepcopy.go
make generate

# 生成 config/crd/bases 和 config/rbac/role.yaml
make manifests
```

7.4.1.5.4 制作OPERATOR镜像

1. 修改kafka-operator/Dockerfile内容如下：

```
FROM centos:latest
COPY bin/manager /
RUN chmod ug+x /manager
WORKDIR /
```

2. 构建容器镜像命令如下：

```
$ make && make docker-build IMG=kafka-operator:v0.0.1
```

7.4.1.6 安装Controller

7.4.1.6.1 安装KUSTOMIZE

基于模板生成YAML文件，下载kustomize二进制压缩包：https://github.com/kubernetes-sigs/kustomize/releases/download/kustomize/v3.8.1/kustomize_v3.8.1_linux_amd64.tar.gz

```
# 解压并安装
$ tar zxvf kustomize_v3.8.1_linux_amd64.tar.gz
$ chmod +x kustomize
$ mv kustomize /usr/local/bin/
$ kustomize version
{Version:kustomize/v3.8.1 GitCommit:0b359d0ef0272e6545eda0e99aacd63aef99c4d0 BuildDate:2020-07-
16T00:58:46Z GoOs:linux GoArch:amd64}
```

7.4.1.6.2 安装CONTROLLER-GEN

构建控制器所使用的Go语言controller-runtime库。

1. 下载源码controller-tools-0.3.0。
2. 构建二进制并安装。

```
$ unzip controller-tools-0.3.0.zip
$ go build -a -o controller-gen cmd/controller-gen/main.go
$ mv controller-gen /usr/local/bin/
$ controller-gen --version
Version: (devel)
```

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com