



中间件云原生化实践分享

相较于传统云计算，云原生改变了什么？

演讲人：王星锦



前世

传统云计算时代，中间件的云上运作方式以及相应的优缺点。



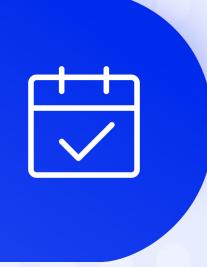
今生

基于K8s的中间件云原生化是如何实现的，相关模式与思想，技术实践和产品化



回顾

云原生实践中走过的弯路，如何能够做的更好、更加的“云原生”



展望

弹性、高可用、自动化的高级特性展望



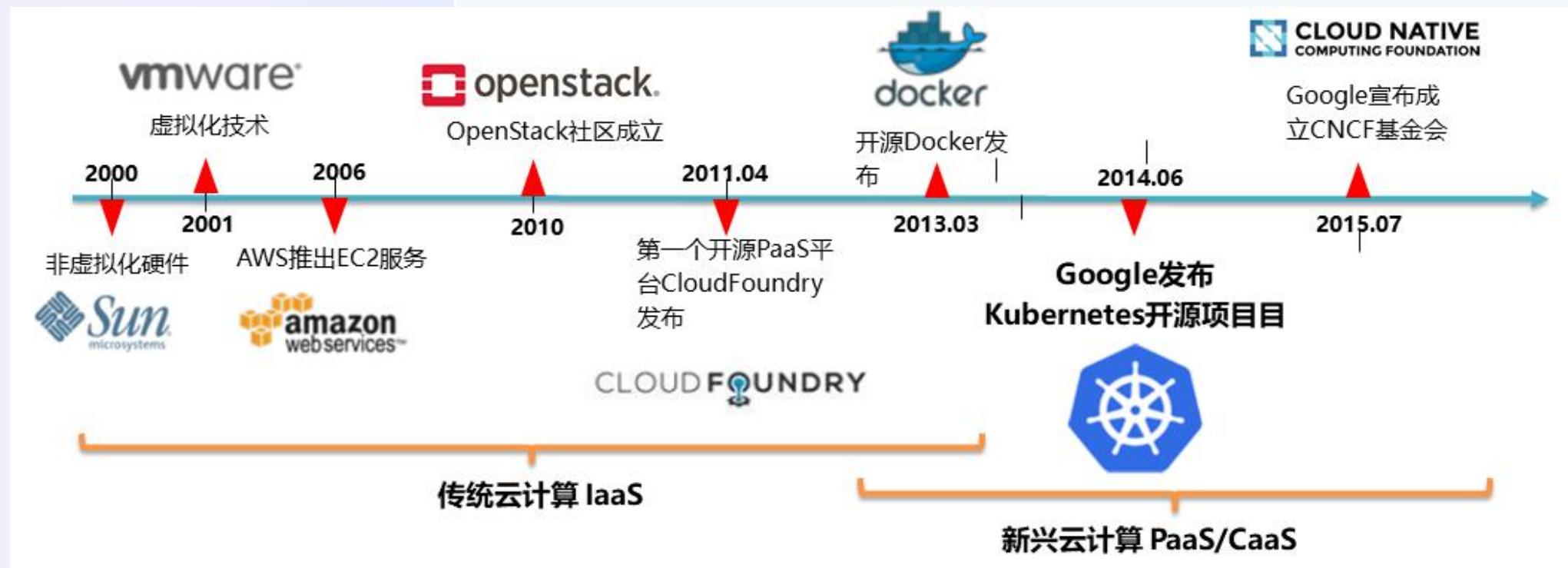
01

前世

传统云计算时代，中间件的云上运作方式以及相应的优缺点。

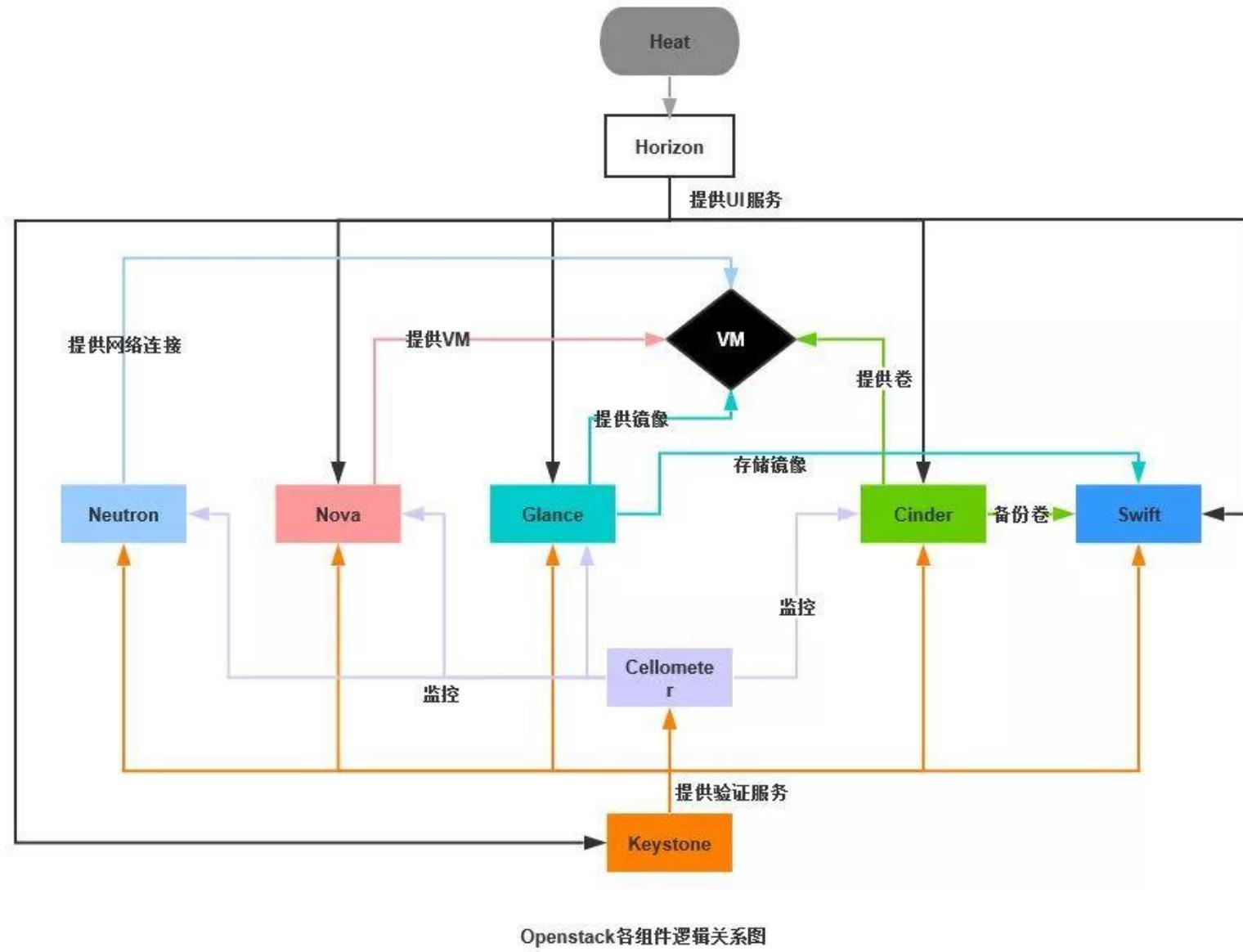
前世-传统云计算

- 云计算发展与演进



前世-传统云计算

- 传统云计算，可以简单理解为我们之前常使用的**云主机**的模式，业界主流的技术栈就是OpenStack
- OpenStack提供多个核心组件，分管计算、网络、存储、镜像、认证等多个功能模块。
- 业务使用云主机时，之前在物理服务器的使用经验大多可以直接迁移，没有认知上的代沟。

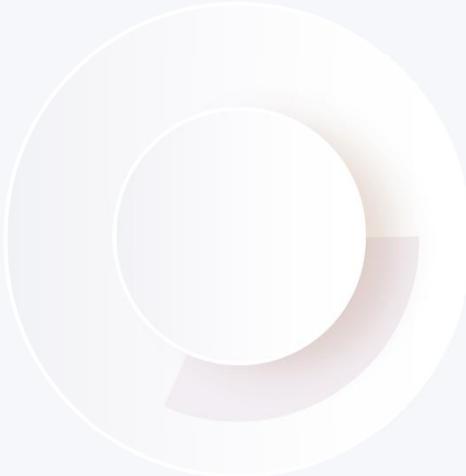


前世-传统云计算

- NCR (Netease Cloud Redis) 在集团内部私有云和公有云中被广泛使用。
- NCR的管控服务本质上是一个和管理Redis实例生命周期的Web服务，通过与OpenStack API、云公共API、主机Agent交互实现Redis在云主机的标准部署。

The screenshot shows the NCR (Netease Cloud Redis) management interface. On the left, a sidebar lists various cloud services: NAT 网关, 负载均衡, Endpoint, 弹性公网 IP, 弹性内网 IP, 私有 DNS, 高速通道, RDS, DDB, and Redis. The Redis icon is highlighted. The main panel is titled "Redis" and displays a single instance: "redis-oneshard" located in "可用区 A", status "运行中 (Normal)", type "主从版", engine "redis 2.8", and created at "今天 23:08". There are buttons for "更改配置" (Change Configuration), "删除" (Delete), and "转包年包月" (Switch to Annual Plan). The top right corner shows user information "ncrjdbtest".

前世-传统云计算



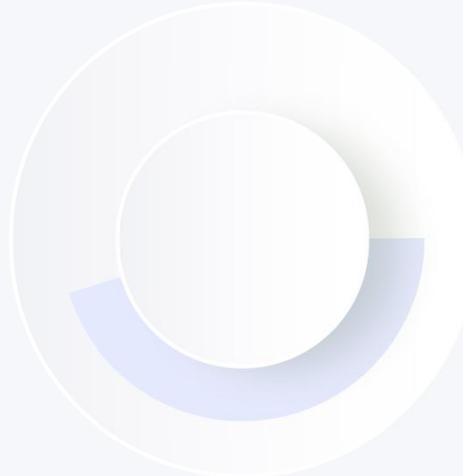
云主机管理较重

管控服务中针对云主机的管理API繁多，任务流程繁杂，任务异常锁定、回滚等场景很难全面覆盖设计



自动化缺乏标准

缺乏自动化任务标准、指令下发、服务发现依赖额外组件，需要自行组合实现。



“重量级选手”

无论是OpenStack的架构、部署、配置还是虚拟化层面，在现在看来它都显得过于沉重、私有化交付难度较大。



02

今生

基于K8s的中间件云原生化是如何实现的，相关的模式与思想，技术实践和产品化

【今生-中间件云原生化】



什么是云原生？

CNCF（云原生计算基金会）的定义：

[CNCF对云原生的定义](#)

云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明式API。

I 今生-中间件云原生化

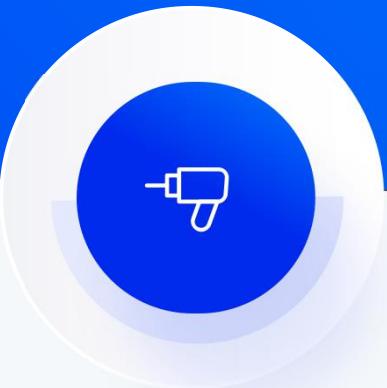
根据实践经验，我自行归纳出的，对云原生的理解姿势



云原生不是具体的技术框架，而是一套技术体系和方法论。



生于云上，长于云上，面向云和弹性设计，设计之初就要考虑充分利用云的弹性和自动化。



云原生系统容错性要高，要对动态的变化持开放态度，从担心故障到“视故障为常态”。

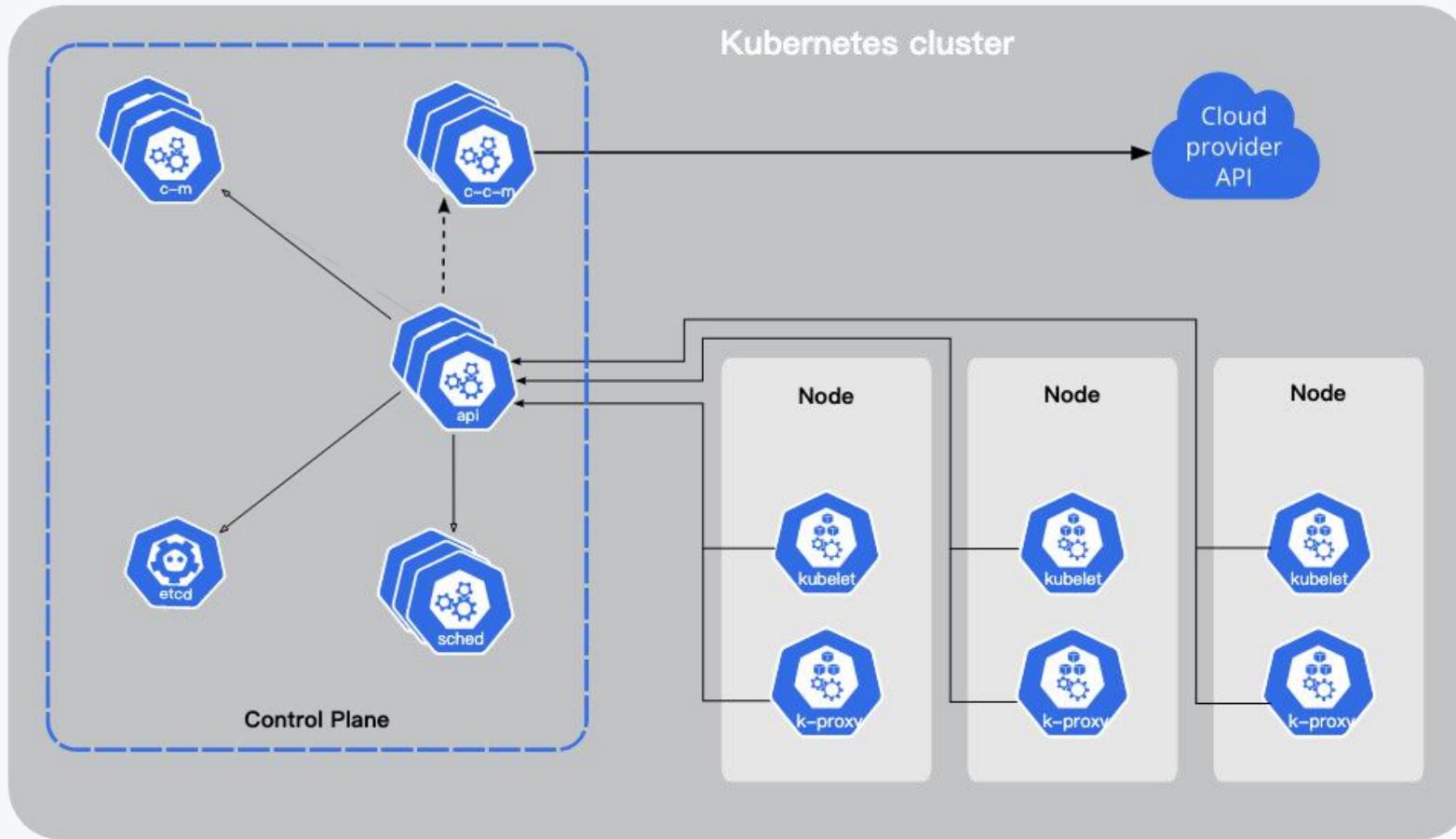
■ 今生-中间件云原生化

Kubernetes 是云原生实践的关键所在，已经是容器编排领域的事实标准，怎么强调都不为过，有一个被越来越多人认可的说法：
Kubernetes是云原生时代的Linux。



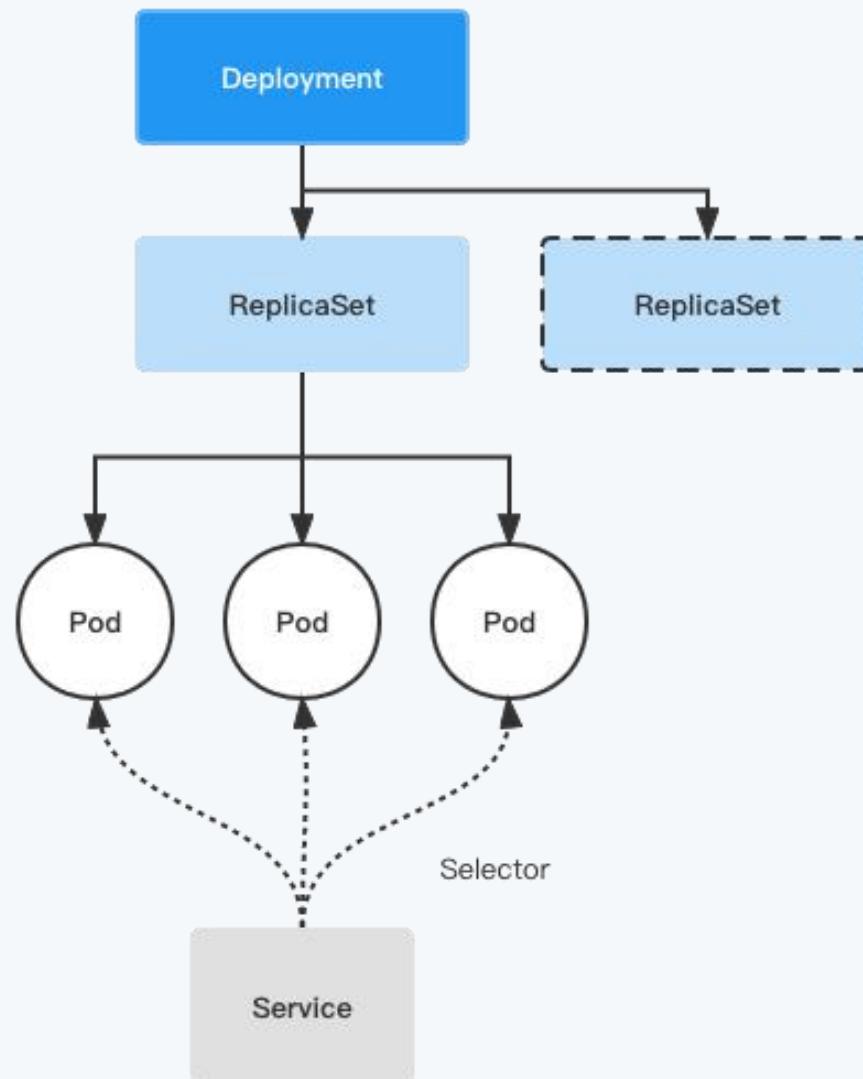
今生-中间件云原生化

K8s架构图



今生-中间件云原生化

- 以K8s中常用的原生资源 Deployment举例
- 资源声明由YAML文件表述，spec字段标识资源的特征值，status字段标识资源的状态
- Deployment的生命周期管理由K8s内置的Deployment Controller控制，原生资源类型都有自己的Controller控制，这个过程被称为调谐（Reconcile）



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

【今生-中间件云原生化】

云原生实践分布不均匀，在K8s、微服务和服务网格的加持下，业务无状态应用接入已经相当成熟和便利。但是对于基础设施的中间件等有状态服务，仍需要特化开发。



今生-中间件云原生化

稳定的网络标识
有状态应用节点间需要互相通信



持久化存储卷
Pod本身状态易失
需要额外的持久化存储



有状态应用的需求

启动顺序要求
由于主备关系，要求启动有序号关系

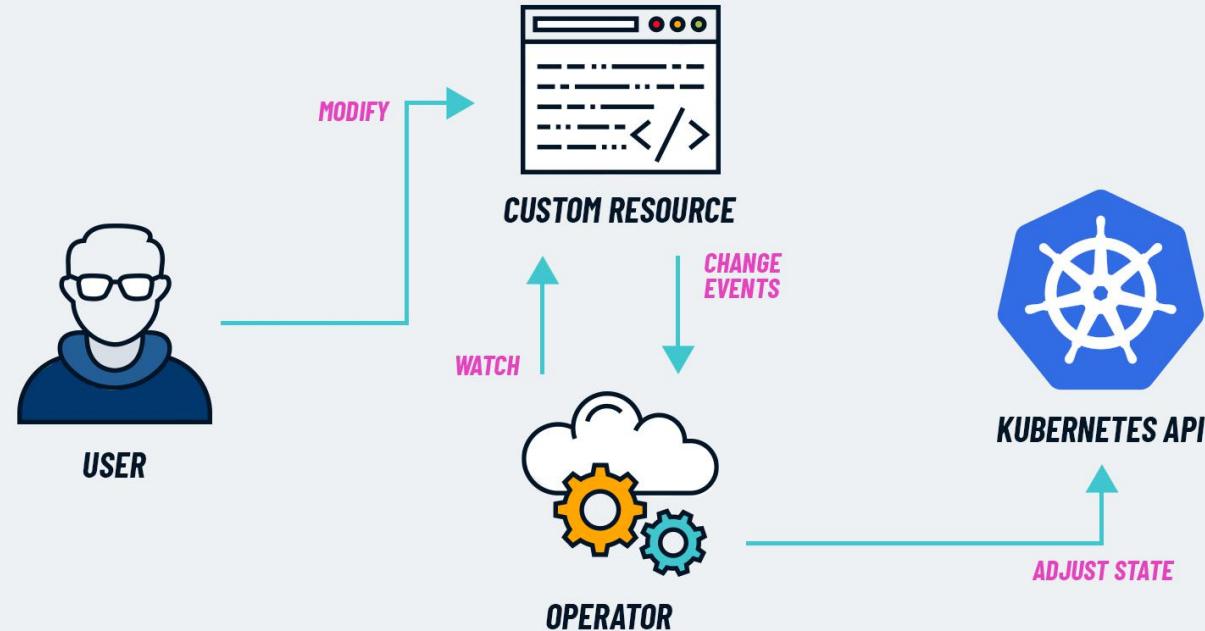


拓扑关系组合
涉及到集群组建、节点发现功能，往往需要第三方介入



今生-中间件云原生化

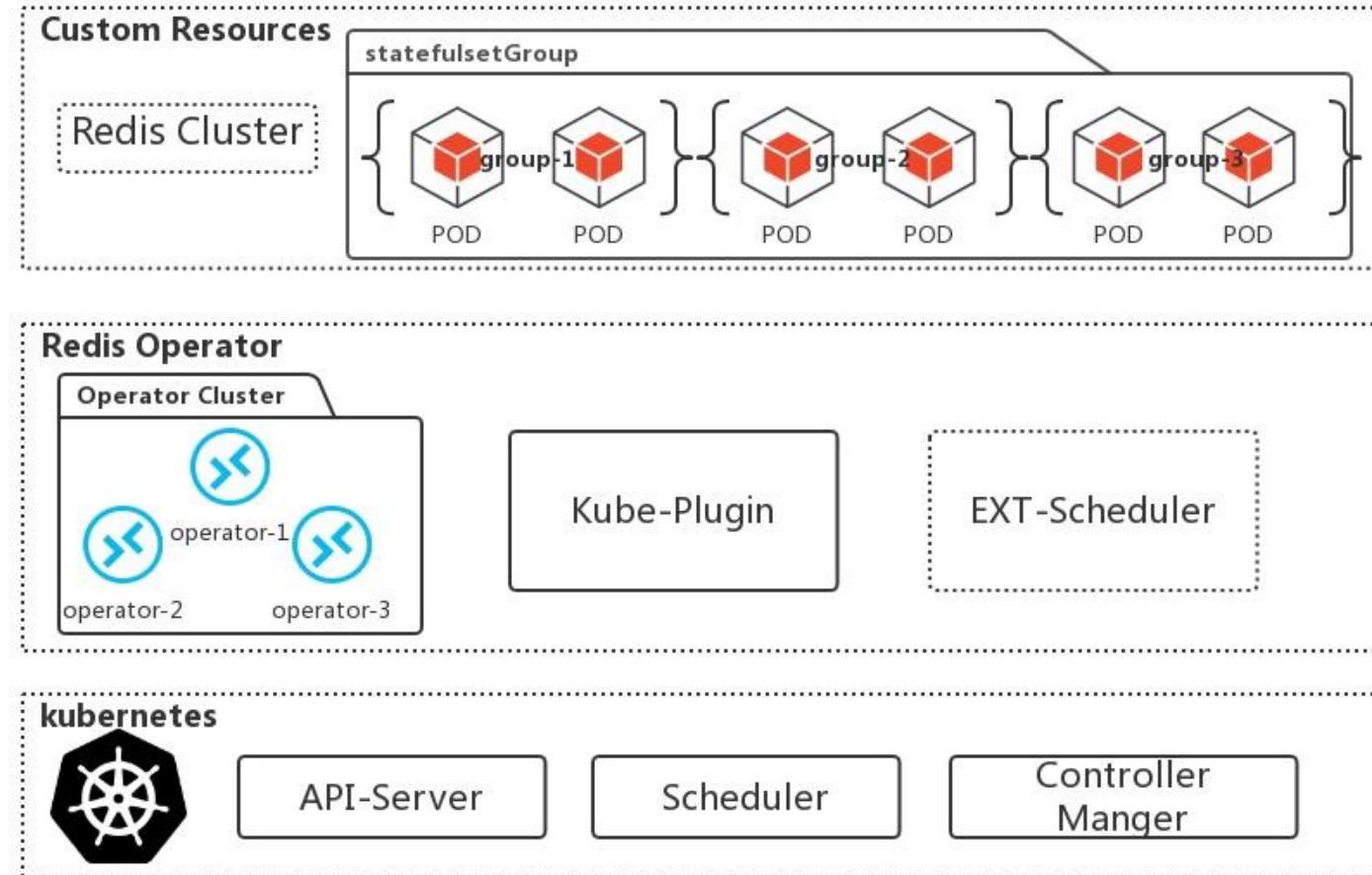
- Operator = CRD + 自定义 Controller
- 应该理解为对K8s的资源和控制器扩展，遵循K8s的设计模式。
- Operator旨在将领域性的运维知识编写成代码融入其中。
- 人工运维依赖于我们脑中的知识，我们应该将这些知识编写成代码。通过明确的指标去做判断和运维



The goal of an Operator is to put operational knowledge into software.

今生-中间件云原生化

- 以Redis Operator举例，架构介绍
- 生命周期、运维能力、高可用调度，全都充分运用K8s原生资源能力或者按照K8s的方式做扩展。
- 一个场景：Master实例宕机后的处理
 - 是否主从切换成功
 - 新的主实例负载是否可控
 - Master内存空闲是否足够
 - 执行恢复



今生-中间件云原生化

- 以Redis集群水平扩容为例，描述Operator处理流程
- 没有前置状态依赖，始终是向期望状态转变。

Redis CR YAML示例

```
apiVersion: ncr.netease.com/v1alpha1
kind: NcrCluster # 资源类型，在CRD中所定义
metadata:
  name: cluster-redis # CR名称
  namespace: ns # CR所在的namespace
spec:
  availableZones: # 可用区列表，支持单、多AZ
    - azName: az1
  configFile: ncr-cluster-configmap # 默认的配置参数模板
  master: 3 # 分片数
  port: 6379 # 端口号
  version: redis:4.0.14 # Redis引擎版本
  resourceReqs: # K8s标准资源规格描述
    requests:
      cpu: 1
      memory: 2Gi
    limits:
      cpu: 2
      memory: 4Gi
```

调谐逻辑伪代码

```
func Reconcile(redis *Redis) error {
    // 调谐Redis运行依赖的ConfigMap
    ReconcileRedisConfigMap(redis)
    // 调谐Redis运行负载StatefulSet
    ReconcileRedisStatefulSet(redis)
    // 调谐Redis集群拓扑状态
    ReconcileRedisClusterTopology(redis)
    // 调谐Redis访问服务
    ReconcileRedisService(redis)
    ...
}
```

今生-中间件云原生化

核心功能都是中间件的声明周期管理，有什么区别？

- 能力控制方式：命令式API与声明式API，声明式API自描述性强
- 计算资源：VM持久化属性更强，硬件虚拟化，性能损耗较高；容器更为轻量，对动态变化接纳性更好
- 访问模式：传统云依赖外置VIP、负载均衡服务；K8s可以原生自带服务发现，LoadBalancer可以对接云厂商的负载均衡服务。
- 安全性：容器的弱隔离性导致其相较于VM，安全性较差。

Redis CR YAML示例

```
apiVersion: ncr.netease.com/v1alpha1
kind: NcrCluster # 资源类型，在CRD中所定义
metadata:
  name: cluster-redis # CR名称
  namespace: ns # CR所在的namespace
spec:
  availableZones: # 可用区列表，支持单、多AZ
  - azName: az1
  configFile: ncr-cluster-configmap # 默认的配置参数模板
  master: 3 # 分片数
  port: 6379 # 端口号
  version: redis:4.0.14 # Redis引擎版本
  resourceReqs: # K8s标准资源规格描述
    requests:
      cpu: 1
      memory: 2Gi
    limits:
      cpu: 2
      memory: 4Gi
```

调谐逻辑伪代码

```
func Reconcile(redis *Redis) error {
    // 调谐Redis运行依赖的ConfigMap
    ReconcileRedisConfigMap(redis)
    // 调谐Redis运行负载StatefulSet
    ReconcileRedisStatefulSet(redis)
    // 调谐Redis集群拓扑状态
    ReconcileRedisClusterTopology(redis)
    // 调谐Redis访问服务
    ReconcileRedisService(redis)
    ...
}
```

今生-中间件云原生化

商业化产品现状，主流中间件支持，提供前端页面、OpenAPI、监控告警等平台能力。

以Redis举例，中间件主干能力列表：

- 单、多AZ高可用部署支持
- 集群创建、删除
- 完善的故障自愈
- 在线水平扩缩容
- 垂直扩缩容
- 热配置更新
- 基于Prometheus的监控和报警功能
- 备份恢复
- 集群联邦
-

The screenshot shows the Qingshuo Cloud Native platform interface. The top navigation bar includes '轻舟云原生' (Qingshuo Cloud Native), '租户 tenant1' (Tenant), '项目 project1' (Project), '运维管理' (Operations Management), and a '超级管理员' (Super Admin) profile icon.

The left sidebar has a '中间件' (Middleware) category selected, with sub-options: 概览 (Overview), 集群管理 (Cluster Management), ElasticSearch, Kafka, Nginx, RabbitMQ, Redis (selected), RocketMQ, ZooKeeper, and 稳定性管控 (Stability Management).

The main content area is titled 'Redis' and shows the '容器集群' (Container Cluster) tab. It features a table with three Redis clusters:

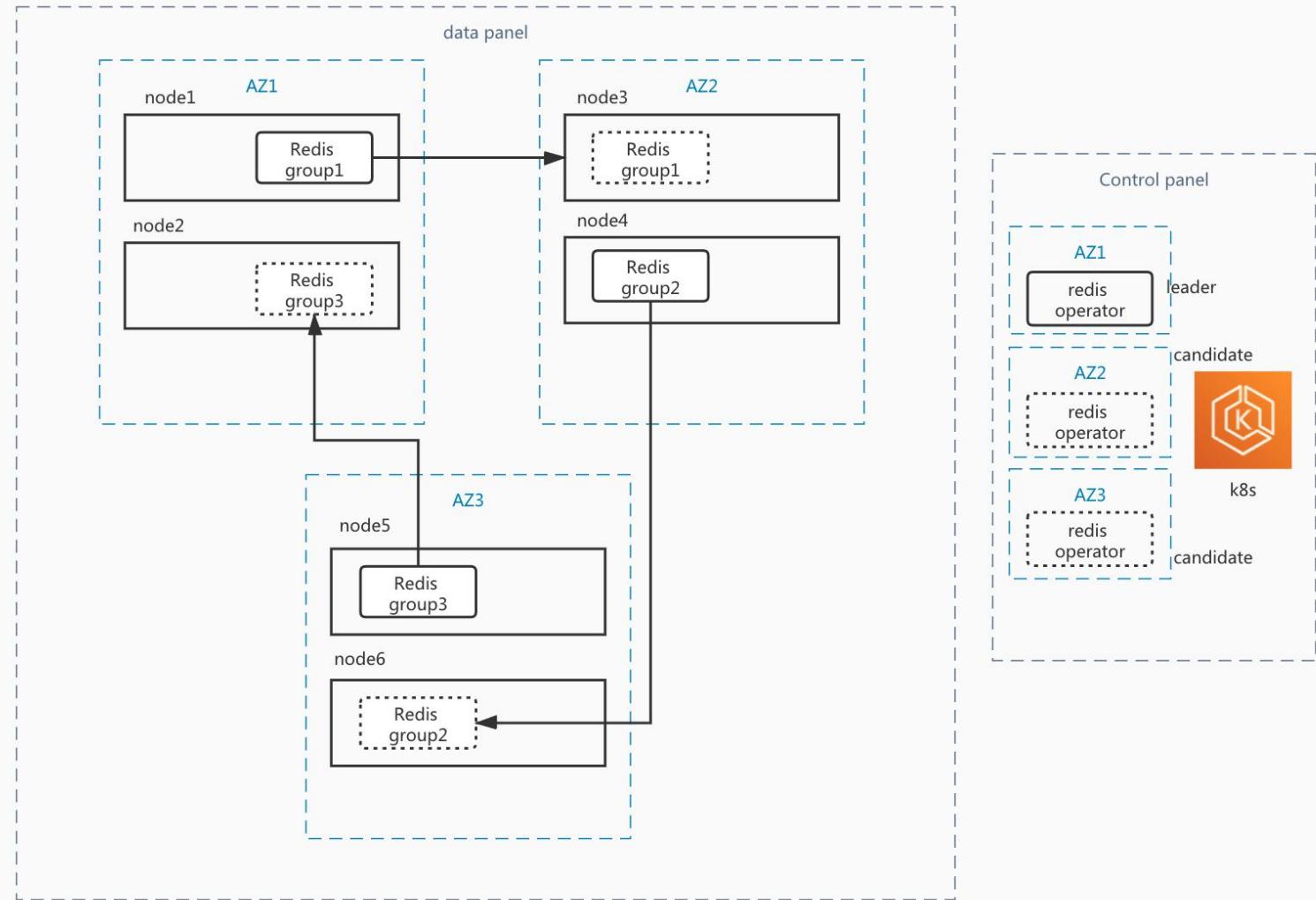
集群名称	集群归属	版本	主/从实例数	内存使用率(已用/总量)	Proxy状态	状态	创建时间	操作
cluster-hsj6	test244-qingzhou-ctrl/ns1	redis:4.0.14	3 / 3	2.52%(15.15MB/0.59GB)	未开启	运行中	2022-12-13 14:27:13	设置 停止 更多
cluster-hsj5	test244-qingzhou-ctrl/ns1	redis:4.0.14	3 / 3	2.52%(15.12MB/0.59GB)	未开启	运行中	2022-12-13 14:14:57	设置 停止 更多
wxj-p1	test244-qingzhou-ctrl/ns1	redis:4.0.14	3 / 3	5.06%(15.17MB/300.00MB)	未开启	运行中	2022-12-13 11:39:55	设置 停止 更多

Table footer: 共 3 条 20条/页 < > 1

今生-中间件云原生化

不同维度的高可用级别

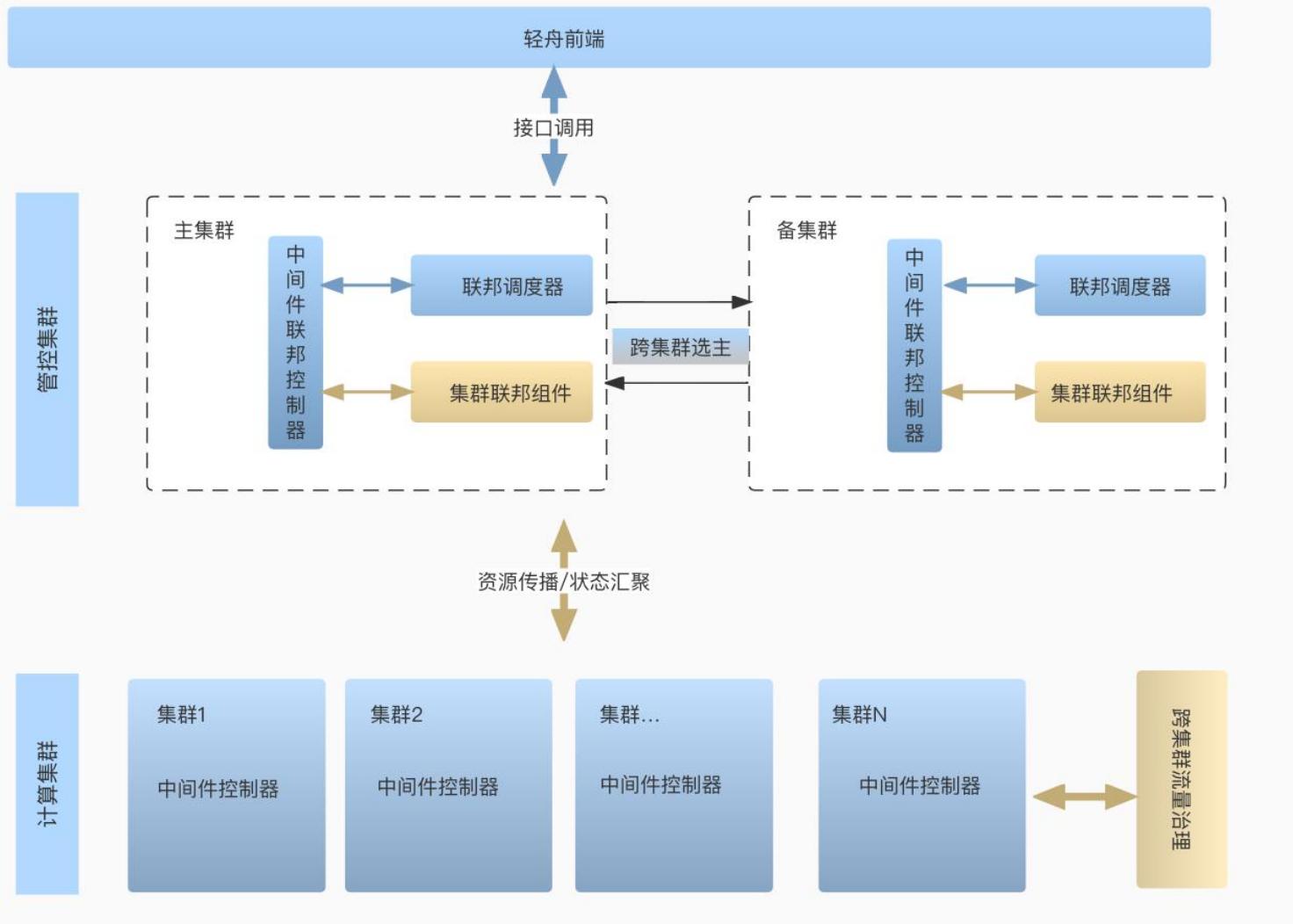
- 单K8s集群单AZ
- 单K8s集群多AZ
- K8s集群联邦



今生-中间件云原生化

不同维度的高可用级别

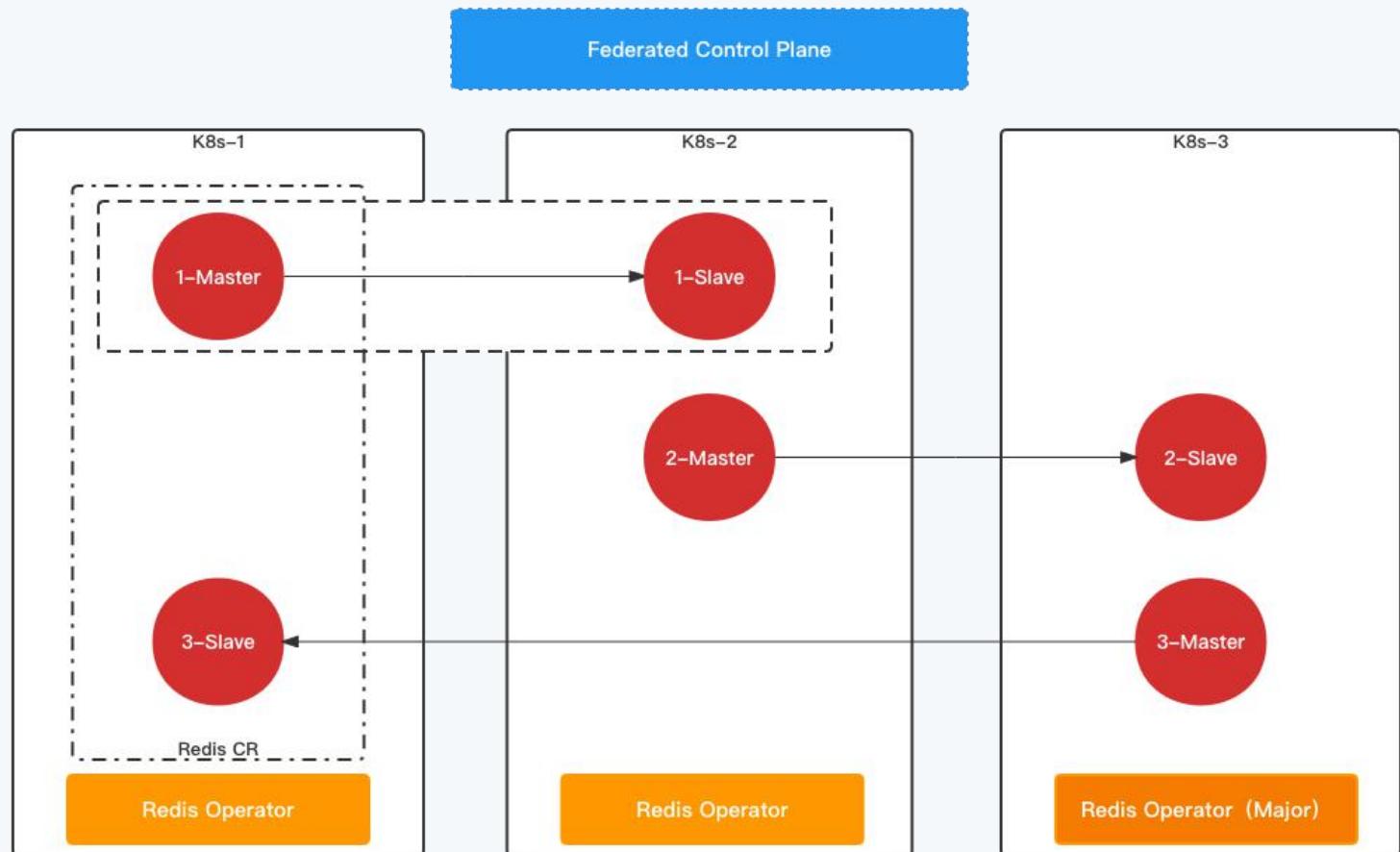
- 单K8s集群单AZ
- 单K8s集群多AZ
- K8s集群联邦



今生-中间件云原生化

集群联邦模式下的Redis集群架构

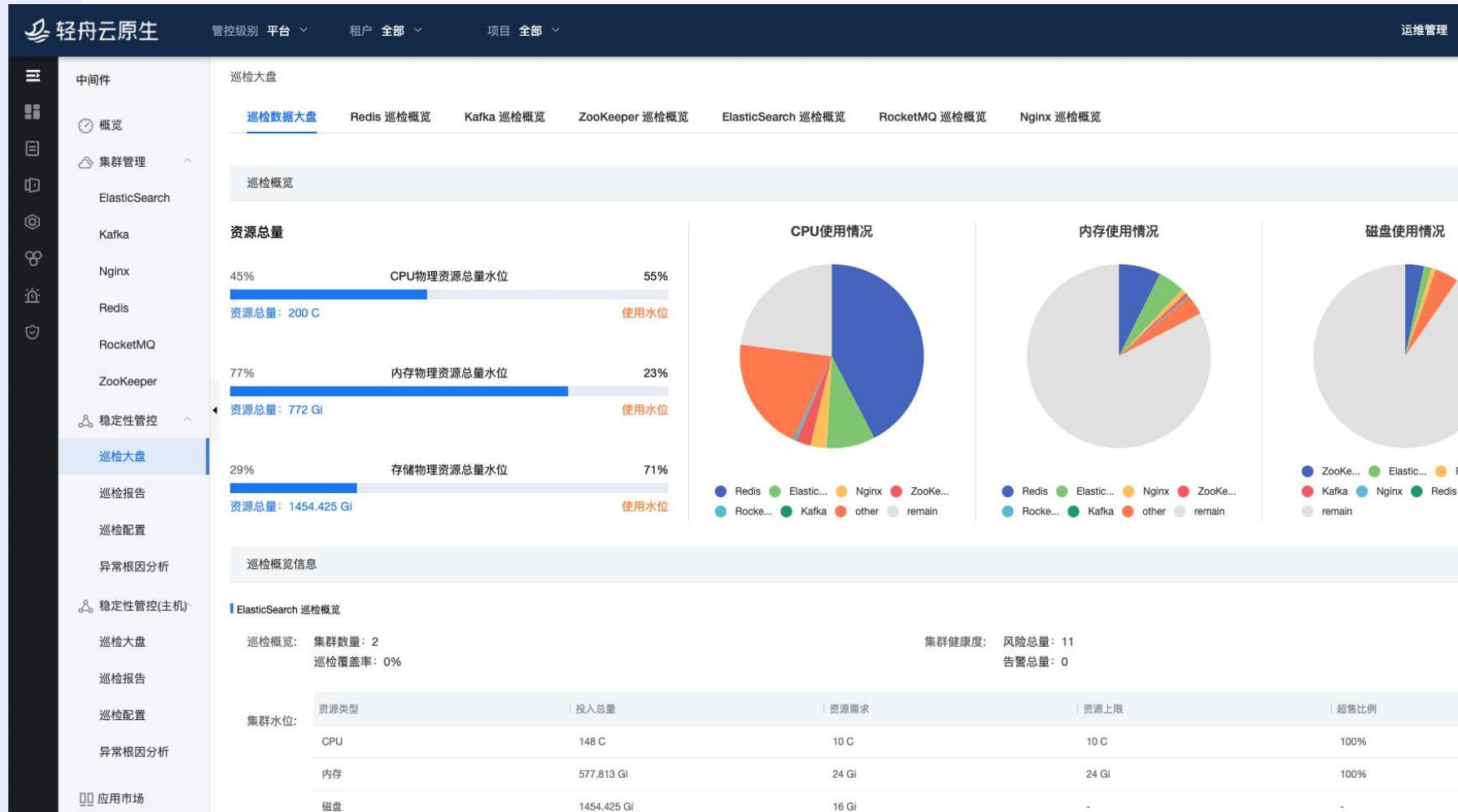
- 每分片主从分布到不同K8s集群
- 各K8s集群资源独立管理
- 故障域隔离清晰直观，不易出错



今生-中间件云原生化

稳定性管控服务

- 预知稳定性隐患，发现风险
- 稳定性巡检
 - 巡检规则库
 - 巡检调度任务
- 根据运维经验，从多维度获取稳定性检查数据
 - 容量水位趋势性问题
 - 资源对象使用合理性
 - 集群配置与架构风险



今生-中间件云原生化

- 快速交付
- 能力开放
- 稳定性保证



Helm部署

Helm类似于CentOS上的 yum 命令，它采用一种名为图表（chart）的打包格式，用以描述一组相关的 K8s 资源文件集合，帮助标准化交付、迭代。



OpenAPI

原生的K8s API对于前端接入，显得较为复杂。我们通过横向服务抽象K8s API为更容易理解的、更符合业务直觉的OpenAPI对外提供。



混沌工程

非功能测试中的故障测试，我们采用混沌实验的方式，使用社区的ChaosMesh 对K8s的负载、网络等随时、随机的注入故障，验证应用的动态容错能力。

今生-中间件云原生化

混沌工程工具ChaosMesh使用介绍

The screenshot shows the Chaos Mesh web interface. At the top, there's a navigation bar with the Chaos Mesh logo and a search bar. Below the header, a sidebar on the left lists various simulation types: 模拟 Pod 故障, 模拟网络故障, 模拟压力场景, 模拟文件 I/O 故障, 模拟 DNS 故障, 模拟时间故障, 模拟 JVM 应用故障, 模拟 Linux 内核故障, 模拟 AWS 故障, 模拟 Azure 故障, 模拟 GCP 故障, and 模拟 HTTP 故障. The "模拟时间故障" option is currently selected. The main content area is titled "新的实验" (New Experiment). It has three tabs: "新的实验" (selected), "加载自" (Load from), and "通过 YAML" (Through YAML). A green box highlights the "选择目标" (Select Target) step. The "填写实验信息" (Fill in experiment information) section includes fields for "范围" (Namespace Selector) and "基本信息" (Basic Information) like "名称" (Name). There are also sections for "高级选项" (Advanced Options) and "运行" (Run) settings like "持续运行" (Continuous Run) and "持续时间" (Duration). A "提交" (Submit) button is at the bottom right. A note at the bottom says "通过勾选 Pod 来进一步限制范围" (Further limit the scope by selecting Pods).

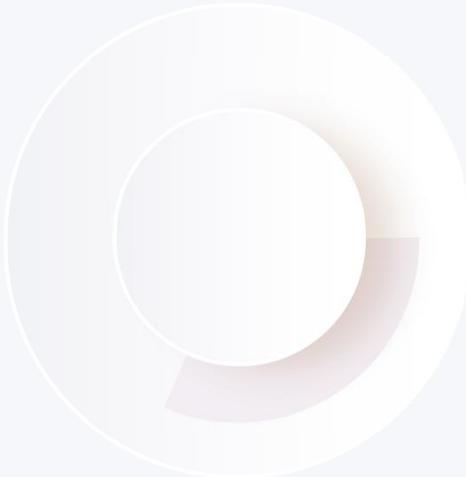


03

回顾

云原生实践中走过的弯路，如何能够做的更好、更加的“云原生”

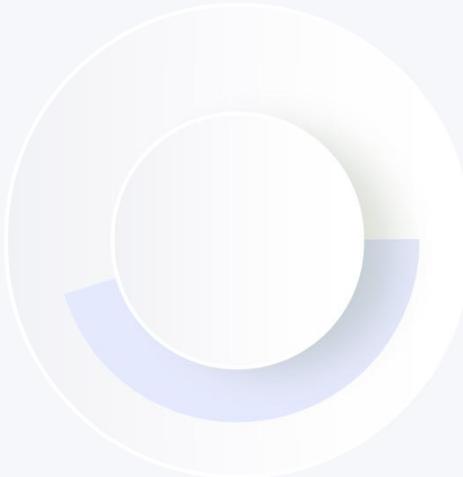
回顾



早期设计不够云原生
云主机运维思路根深蒂固，最早做Redis Operator时有很多设计没有充分运用K8s的能力，后续功能设计不够整洁。



权限控制细化
这是在基于K8s开发时容易忽略的问题，要尽可能避免特权容器的使用，基于K8s RBAC做好细粒度的权限管理。



深入K8s原生特性
引入一个功能时，首先要思考：K8s是否有原生资源和特性可以直接支持？在尽可能的运用原生特性后，我们如何最小化介入？



04

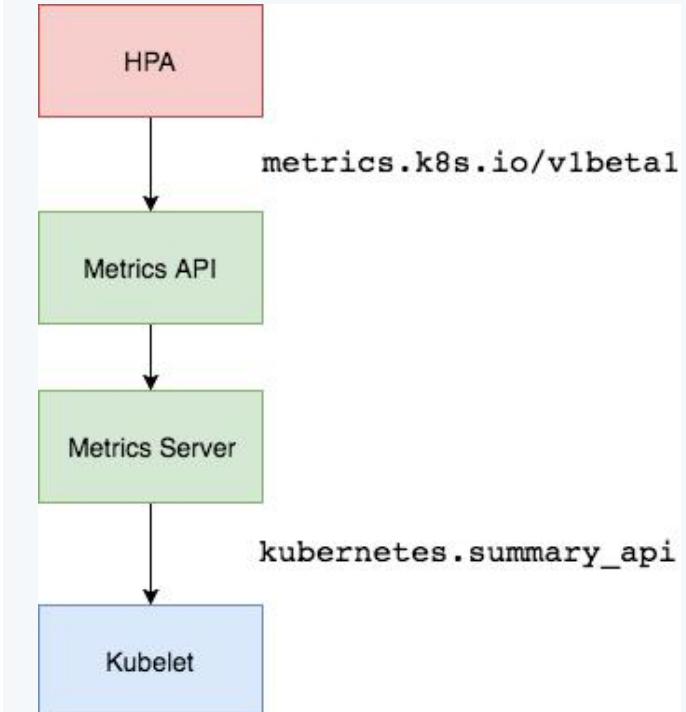
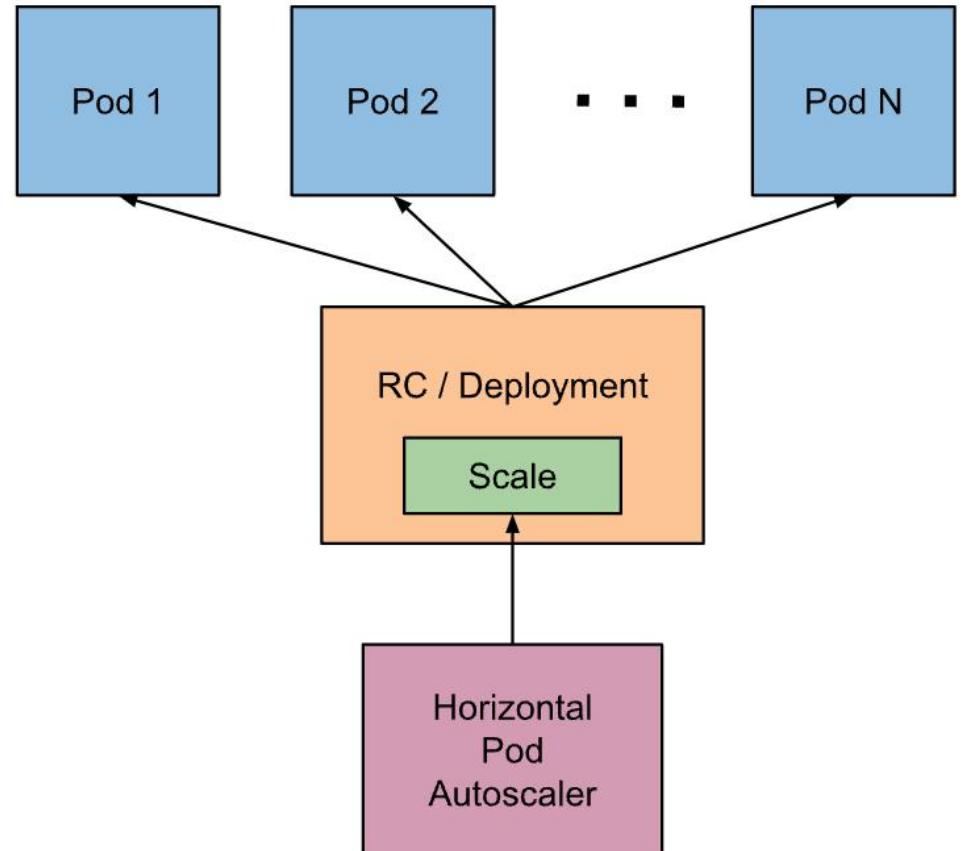
展望

弹性、高可用、自动化的高级特性展望

展望

从K8s的新特性入手，分析是否可以迁移到我们的产品功能设计

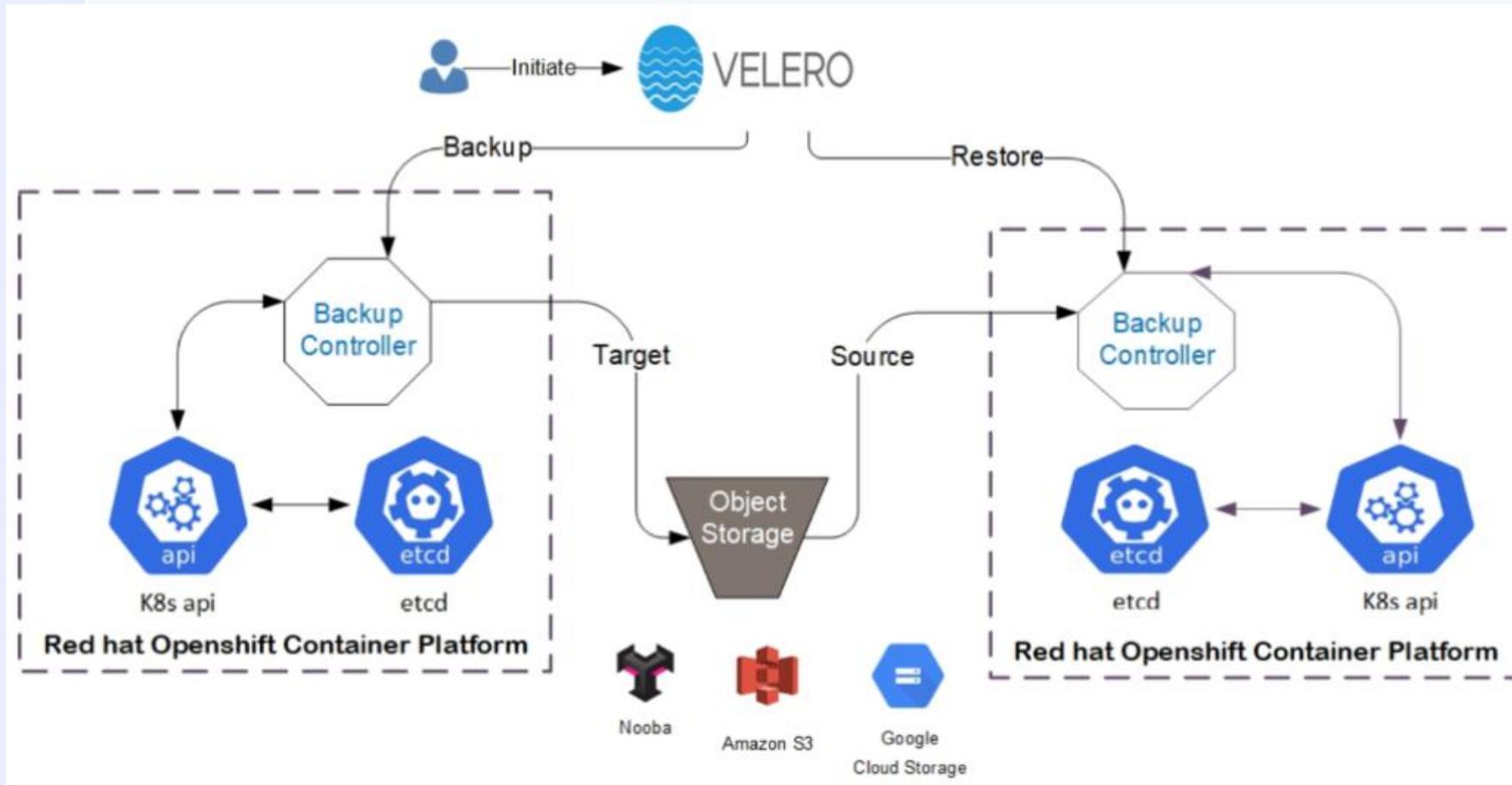
- HPA (Pod水平自动伸缩)，根据负载自动调节副本数
- 有状态应用可以自定义指标，在Operator中引入类似逻辑
- 水位调度



展望

灾备能力再加强

- 中间件单元化和异地多活
- 跨K8s集群的备份恢复



展望

目前遇到的问题，未来需要考虑的情况



K8s API迭代速度很快，每季度发布的版本间就可能会出现API字段不兼容，针对这种情况Operator如何处理？



商业化产品难免产生定制化功能，如何把定制功能与标准功能做到松耦合？



用户大量使用中间件后，如果开源组件出现漏洞问题，如何支持批量升级运维的能力，并保证可观测性？



谢谢观看



关注了解更多