阿里云 × CLOUD NATIVE COMPUTING FOUNDATION

云原生技术公开课

第 24 讲

Kubernetes API 编程利器

Operator 和 Operator Framework

夙兴 阿里巴巴高级开发工程师

关注“阿里巴巴云原生”公众号
获取第一手技术资料

1

Operator 概述

2

Operator framework
实战

3

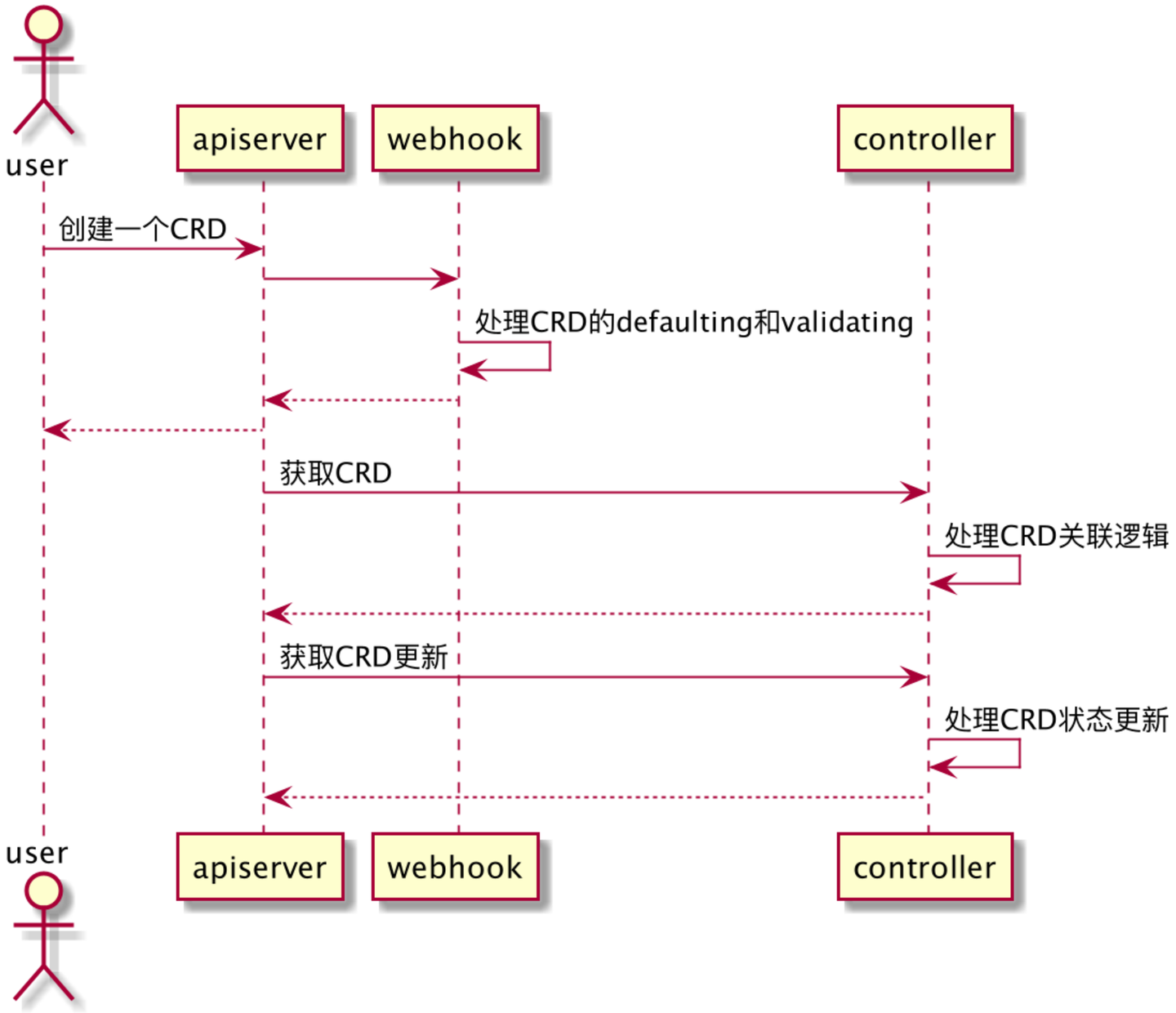
工作流程

基本概念

- **CRD：**Custom Resource Definition，允许用户自定义 kubernetes 资源
- **CR：**Custom Resource，CRD 的具体实例
- **webhook：**webhook 关联在 apiserver 上，是一种 HTTP 回调，一个基于 web 应用实现的 webhook 会在特定事件发生时把消息发送给特定的 URL 。用户一般可以定义两类 webhook，分别是 mutating webhook（变更传入对象） 和 validating webhook（传入对象校验）
- **工作队列：**controller 核心组件，controller 会监控集群内关注资源对象的变化，并把相关对象的事件（动作和key），存储于工作队列中
- **controller：**controller 是监测集群状态变化，并据此作出相应处理的控制循环。它关联一个工作队列，循环处理队列内容。每一个 controller 都试图把集群状态向预期状态推动，只是关注的对象不同，如 replicaset controller 和 endpoint controller
- **operator：**operator 是描述、部署和管理 kubernetes 应用的一套机制，从实现上来说，**operator = CRD + webhook + controller**

常见operator工作模式

1. 用户创建一个 CRD
2. apiserver 转发请求给 webhook
3. webhook 负责 CRD 的缺省值配置和配置项校验
4. controller 获取到新建的 CRD，处理“创建副本”等关联逻辑
5. controller 实时检测 CRD 相关集群信息并反馈到 CRD 状态



1

Operator 概述

2

Operator framework
实战

3

工作流程

Operator framework 概述

- Operator framework 给用户提供了 webhook 和 controller 框架，包括消息通知、失败重新入队等等，开发人员仅需关心被管理应用的运维逻辑实现
- 主流 Operator framework 项目
  - kubebuilder：https://github.com/kubernetes-sigs/kubebuilder
  - operator-sdk：https://github.com/operator-framework/operator-sdk
  - 两者没有本质上的区别，都是使用的 controller-tools 和 controller-runtime。细节上 kubebuilder 相应的测试、部署、代码生成脚手架更完善，如 Makefile 和 Kustomize 等工具的集成；operator sdk 则支持与 ansible operator、Operator Lifecycle Manager 的集成

kubebuilder 实战 - step 1. 初始化

- 案例描述：SidecarSet 是开源社区的一个 operator 实现，它负责给 pod 插入 sidecar 容器，并负责更新 sidecar 状态
  - 注1：切到 e8d836227eb979b5874841c45ab15cc739e7a5d9 commit 学习，这个版本功能较为简单，方便学习
  - 注2：kubebuilder 当前处于2.0.1版本，本次实战仍使用1.x版本

- 新建一个gitlab项目，运行：kubebuilder init --domain=kruise.io
  - 参数解读：domain 指定了后续注册 CRD 对象的 Group 域名
  - 效果解读：拉取依赖代码库、生成代码框架、生成Makefile/Dockerfile等工具文件

kubebuilder 实战 - step 1. 初始化结果

注1：为了方便展示，实际生成文件做了部分删减

```
git show 01e89868bd31bdc9ef71f5f34cf5c3a40faf5f85 | grep "+++ b”
+++ b/Dockerfile
+++ b/Gopkg.lock
+++ b/Gopkg.toml
+++ b/Makefile
+++ b/PROJECT
+++ b/cmd/manager/main.go
+++ b/config/default/kustomization.yaml
+++ b/config/rbac/auth_proxy_role.yaml
+++ b/hack/boilerplate.go.txt
+++ b/pkg/apis/apis.go
+++ b/pkg/controller/controller.go
+++ b/pkg/webhook/webhook.go
+++ b/vendor/cloud.google.com/go/CHANGES.md
```

kubebuilder 实战 - step 2. 创建 API（CRD）

运行：

kubebuilder create api --group apps --version v1alpha1 --kind SidecarSet --namespaced=false

参数解读：

- group+前面的domain，即为此CRD的Group：apps.kruise.io
- version一般三种，按社区标准：
  - v1alpha1：此 api 不稳定，CRD 可能废弃、字段可能随时调整，不要依赖
  - v1beta1：api 已稳定，会保证后向兼容，特性可能调整
  - v1：api 和特性都已稳定
- kind：此 CRD 的类型，类似于社区原生的 Service 的概念
- namespaced：此CRD是全局唯一还是namespace唯一，类似 node 和 pod

效果解读：

- **生成了 CRD 和 controller 的框架**，后面需要手工填充代码

kubebuilder 实战 - step 2. 创建 API（CRD）结果

```
git show f1c8dcf90129b6cf1013797e84754f018aa84e74 | grep "+++ b”
+++ b/config/crds/apps_v1alpha1_sidecarset.yaml
+++ b/config/rbac/rbac_role.yaml
+++ b/config/samples/apps_v1alpha1_sidecarset.yaml
+++ b/pkg/apis/addtoscheme_apps_v1alpha1.go
+++ b/pkg/apis/apps/group.go
+++ b/pkg/apis/apps/v1alpha1/doc.go
+++ b/pkg/apis/apps/v1alpha1/register.go
+++ b/pkg/apis/apps/v1alpha1/sidecarset_types.go
+++ b/pkg/apis/apps/v1alpha1/sidecarset_types_test.go
+++ b/pkg/apis/apps/v1alpha1/v1alpha1_suite_test.go
+++ b/pkg/apis/apps/v1alpha1/zz_generated.deepcopy.go
+++ b/pkg/controller/add_sidecarset.go
+++ b/pkg/controller/sidecarset/sidecarset_controller.go
+++ b/pkg/controller/sidecarset/sidecarset_controller_suite_test.go
+++ b/pkg/controller/sidecarset/sidecarset_controller_test.go
```

kubebuilder 实战 - step 3. 填充 CRD

1. 生成的 CRD 位于：pkg/apis/apps/v1alpha1/sidecarset_types.go，按需修改
   - 因为 code generator 依赖注释生成代码，有时需要**调整注释**：
     - +genclient:nonNamespaced：生成非 namespace 对象
     - +kubebuilder:subresource:status：生成 status 子资源
     - +kubebuilder:printcolumn:name="MATCHED",type="integer", JSONPath=".status.matchedPods",description="xxx"：kubectl get sidecarset 后续展示相关
   - 需要**填充字段**：
     - SidecarSetSpec：填充 CRD 描述信息
     - SidecarSetStatus：填充 CRD 状态信息
2. 填充完运行 make 重新生成代码即可

* 注：研发人员不需要参与 CRD 的 grpc 接口、编解码等 Controller 底层实现

kubebuilder 实战 - step 3. 填充 CRD 结果

```
type SidecarSetSpec struct {
    Selector     *metav1.LabelSelector
    Containers []SidecarContainer
}
```

```
type SidecarSetStatus struct {
    MatchedPods int32
    UpdatedPods int32
    ReadyPods    int32
}
```

完整内容可移步：[sidecarset 定义]

kubebuilder 实战 - step 4. 生成 webhook 框架

1. 生成 mutating webhook，运行：

```
kubebuilder alpha webhook --group apps --version v1alpha1 --kind SidecarSet --type=mutating —operations=create
kubebuilder alpha webhook --group core --version v1 --kind Pod --type=mutating --operations=create // 生成 Pod 资源对应的 webhook
```

2. 生成 validating webhook，运行：

```
kubebuilder alpha webhook --group apps --version v1alpha1 --kind SidecarSet --type=validating --operations=create,update
```

参数解读：

- group/kind 描述需要处理的资源对象
- type 描述需要生成哪种类型的框架
- operations 描述关注资源对象的哪些操作

效果解读：

- 生成了 webhook 的框架，后面需要手工填充代码

kubebuilder 实战 - step 4. 生成 webhook 框架结果

```
git show c05fa7f4cd546fb6de9ab6e414b70669572a4daa | grep "+++ b”
+++ b/pkg/webhook/add_default_server.go
+++ b/pkg/webhook/default_server/add_mutating_pod.go
+++ b/pkg/webhook/default_server/add_mutating_sidecarset.go
+++ b/pkg/webhook/default_server/add_validating_sidecarset.go
+++ b/pkg/webhook/default_server/pod/mutating/create_webhook.go
+++ b/pkg/webhook/default_server/pod/mutating/pod_create_handler.go
+++ b/pkg/webhook/default_server/pod/mutating/webhooks.go
+++ b/pkg/webhook/default_server/server.go
+++ b/pkg/webhook/default_server/sidecarset/mutating/create_webhook.go
+++ b/pkg/webhook/default_server/sidecarset/mutating/sidecarset_create_handler.go
+++ b/pkg/webhook/default_server/sidecarset/mutating/webhooks.go
+++ b/pkg/webhook/default_server/sidecarset/validating/create_update_webhook.go
+++ b/pkg/webhook/default_server/sidecarset/validating/sidecarset_create_update_handler.go
+++ b/pkg/webhook/default_server/sidecarset/validating/webhooks.go
```

kubebuilder 实战 - step 5. 填充 webhook

生成的 webhook handler 位于：

- pkg/webhook/default_server/sidecarset/mutating/xxx_handler.go
- pkg/webhook/default_server/sidecarset/validating/xxx_handler.go
- pkg/webhook/default_server/pod/mutating/xxx_handler.go

需要改写、填充的一般有：

- **是否需要注入 K8s client**：webhook 框架会传入需要处理的资源对象，如果在此之外还需要访问其它资源才能完成工作，按照框架内注释修改以注入 client，如上面的 pod/mutating
- **填充 webhook 关键方法 mutatingSidecarSetFn 或 alidatingSidecarSetFn**（待操作资源对象指针已经传入，我们直接调整该对象属性即可完成 hook 的工作）

kubebuilder 实战 - step 5. 填充 webhook 结果

```
func (…) mutatingSidecarSetFn(… {
    setDefaultSidecarSet(obj)
}

func setDefaultSidecarSet(… {
    for i := range containers {
        setDefaultContainer(…
    }
}
```

完整内容可移步：

- sidecarset mutating
- sidecarset mutating
- pod mutating

```
func (…) validatingSidecarSetFn(… {
    allErrs := validateSidecarSet(obj)
    if len(allErrs) != 0 {
        return false, "", allErrs.ToAggregate()
    }
    return true, "allowed to be admitted", nil
}

func validateSidecarSet(… {
    allErrs := genericvalidation.ValidateObjectMeta(…
    allErrs = append(allErrs, validateSidecarSetSpec(…
    return allErrs
}
```

kubebuilder 实战 - step 6. 填充 controller

生成的 controller 框架位于：pkg/controller/sidecarset/sidecarset_controller.go

- **修改权限注释**：框架会自动生成形如 // +kubebuilder:rbac:groups=apps,resources=deployments/status,verbs=get;update;patch的注释，该注释最终会生成rbac规则，按需修改
- **增加入队逻辑**：缺省的代码框架会填充 CRD 本身的入队逻辑（如 SidecarSet 对象的增删改都会加入工作队列），如果需要关联资源对象的触发机制（如 SidecarSet 也需关注 pod 的变化），则需手工新增它的入队逻辑
- **填充业务逻辑**：修改 Reconcile 函数，处理工作队列。完成“根据 spec 完成逻辑”和“将逻辑结果反馈回 status”两部分
- 需要注意的是：Reconcile 函数里 return reconcile.Result{}, err 会重新入队

kubebuilder 实战 - step 6. 填充 controller 结果

```
func (…) addPod(… {
    pod, ok := obj.(*corev1.Pod)
    sidecarSets, err := p.getPodSidecarSets(pod)
        for _, sidecarSet := range sidecarSets {
            q.Add(reconcile.Request{
                types.NamespacedName{
                    Name: sidecarSet.Name,
                },
            })
```

完整内容可移步：

- 入队逻辑
- sidecarset controller

```
func (…) Reconcile(… {
    selector := sidecarSet.Spec.Selector
    r.List(context.TODO(), selector, matchedPods)

    for i := range matchedPods.Items {
        pod := &matchedPods.Items[i]
        if util.IsPodActive(pod) && !isIgnoredPod(pod) {
            filteredPods = append(filteredPods, pod)

    status := calculateStatus(sidecarSet, filteredPods)
    r.updateSidecarSetStatus(sidecarSet, status)
```

1

Operator 概述

2

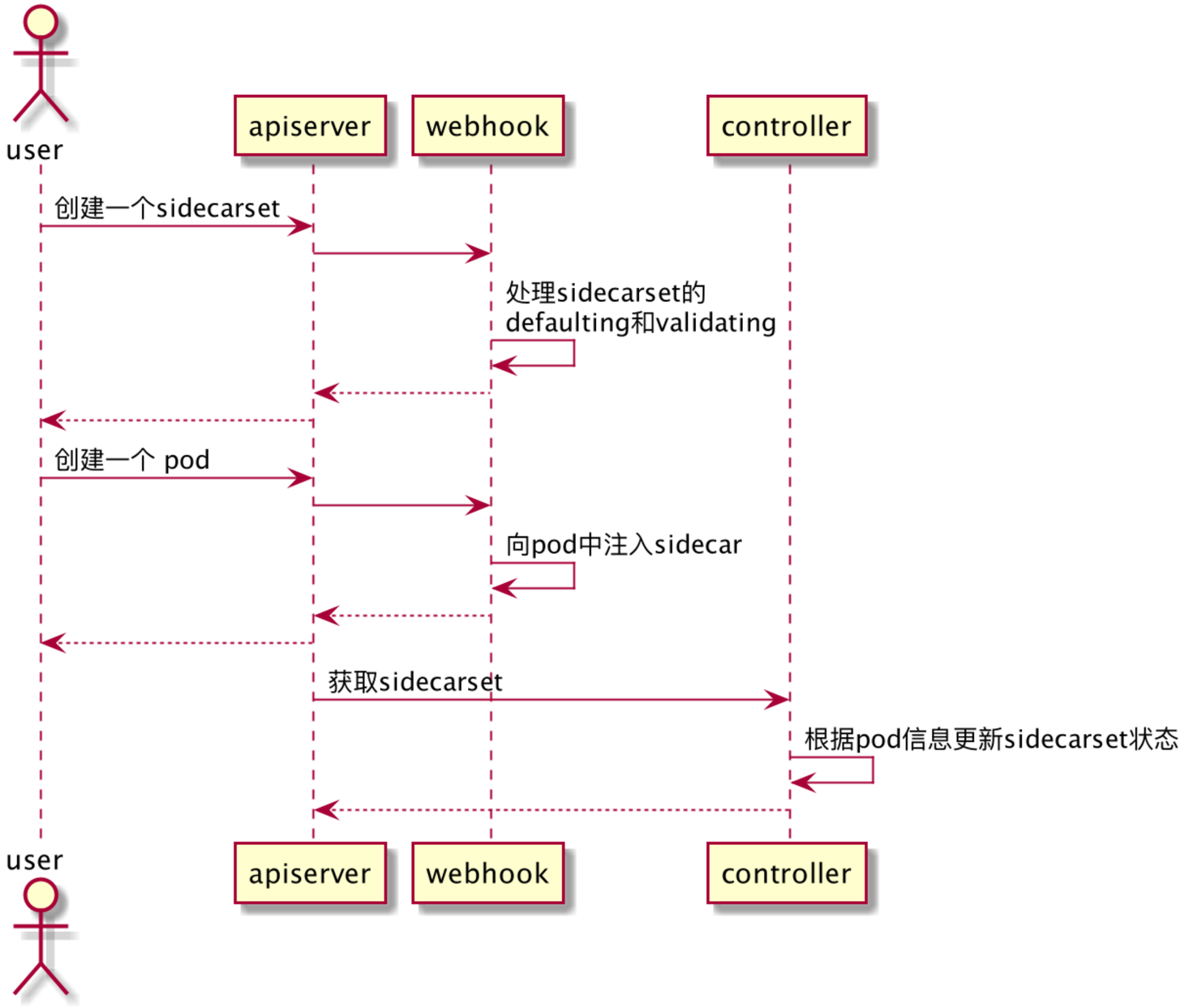
Operator framework
实战

3

工作流程

工作流程

1. 用户创建 sidecarset
2. webhook 负责 sidecars 的缺省值配置和配置项校验
3. 用户创建 pod
4. webhook 负责向 pod 注入 sidecar 容器
5. controller 实时检测 pod 信息，并更新到 sidecarset 状态



关注“阿里巴巴云原生”公众号
获取第一手技术资料