## Example 11-9. The Reconcile method for the AcmeNamespace controller

// controllers/acmenamespace\_controller.go

package controllers

import (

"context"

"github.com/go-logr/logr"

corev1 "k8s.io/api/core/v1"

metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"

"k8s.io/apimachinery/pkg/runtime"

ctrl "sigs.k8s.io/controller-runtime"

"sigs.k8s.io/controller-runtime/pkg/client"

tenancyv1alpha1 "github.com/lander2k2/namespace-operator/api/v1alpha1"

)

...

func (r \*AcmeNamespaceReconciler) Reconcile(req ctrl.Request) (ctrl.Result, error) {

ctx := context.Background()

log := r.Log.WithValues("acmenamespace", req.NamespacedName)

var acmeNs tenancyv1alpha1.AcmeNamespace

r.Get(ctx, req.NamespacedName, &acmeNs)

nsName := acmeNs.Spec.NamespaceName

adminUsername := acmeNs.Spec.AdminUsername

ns := &corev1.Namespace{

ObjectMeta: metav1.ObjectMeta{

Name: nsName,

Labels: map[string]string{

"admin": adminUsername,

},

},

}

if err := r.Create(ctx, ns); err != nil {

log.Error(err, "unable to create namespace")

return ctrl.Result{}, err

}

return ctrl.Result{}, nil

}

...

## 这段代码到底是创建一个CR还是创建一个CRD还是什么东西？这个CR已经被谁创建过了?

ChatGPT

这段代码的目的是创建一个 Namespace 资源，而不是创建一个自定义资源 (CR) 或自定义资源定义 (CRD)。这个 Namespace 的创建是基于一个已经存在的 AcmeNamespace 自定义资源的信息。

在这个例子中，AcmeNamespace 是一个自定义资源 (CR)，它的定义 (CRD) 已经被添加到 Kubernetes 集群中，所以用户可以创建 AcmeNamespace 资源实例。

用户在 Kubernetes 集群中创建一个 AcmeNamespace 资源实例，包含他们想要创建的命名空间的信息（比如命名空间的名称和管理员用户名）。这个 AcmeNamespace 资源被 Kubernetes API 服务器接收并存储。

然后，这段代码中的 Reconcile 方法被 Kubernetes 控制器触发，因为它监视了 AcmeNamespace 资源的创建和更新。Reconcile 方法获取用户创建的 AcmeNamespace 资源的信息，然后使用这些信息创建一个新的 Namespace 资源。

这是 Kubernetes 操作器模式的一个常见示例，用户创建一个自定义资源（在这个例子中是 AcmeNamespace），然后一个控制器（在这个例子中是 AcmeNamespaceReconciler）监视这个自定义资源，当有新的自定义资源被创建或已有的自定义资源被更新时，控制器会响应这些变化并采取相应的动作（在这个例子中是创建一个 Namespace）。

## 这个CR AcmeNamespace的作用，是不是就相当于我实例化了一个Deployment deploymentAAA？

ChatGPT

不完全相同，但你的理解基本上是正确的。当你创建一个Deployment资源时，Kubernetes会使用Deployment控制器根据你的Deployment定义去创建并管理一组Pod。你可以将Deployment视为Pod的一种高级封装，它增加了一些额外的功能，如滚动更新和扩展。

而自定义资源（CR，如AcmeNamespace）和控制器（比如你可能写的一个Operator）的组合，实际上是对Kubernetes API的扩展。这种扩展可以添加新的行为和资源类型到你的Kubernetes集群。当你创建一个AcmeNamespace资源时，你实际上是在告诉你的AcmeNamespace控制器你希望它采取某些动作（例如，在这个例子中，创建一个新的Namespace，然后在其中创建一些资源）。

所以，你可以将创建AcmeNamespace CR看作是给Kubernetes API发送一个请求，告诉它你希望采取某些动作，这些动作是由AcmeNamespace控制器定义的。这和创建Deployment资源有一些相似之处，但是自定义资源和控制器的组合提供了更大的灵活性，因为你可以自己定义它们的行为。

在这段代码中，r.Get(ctx, req.NamespacedName, &acmeNs) 这行的目的是获取 AcmeNamespace 自定义资源的实例。这个资源应该已经存在于 Kubernetes 集群中，因为这个 Reconcile 方法是在 Kubernetes 发现新的 AcmeNamespace 资源或者已存在的 AcmeNamespace 资源发生变化时被触发的。

你正在创建的 Namespace 是由 AcmeNamespace 自定义资源驱动的。AcmeNamespace 资源包含了创建 Namespace 所需的所有信息，例如在这个例子中，它包含了 NamespaceName 和 AdminUsername。

所以，在这里，你正在从 Kubernetes API 服务器获取一个已经存在的 AcmeNamespace 资源，然后从这个资源中提取信息来创建一个新的 Namespace。希望这个答案能够解答你的问题。

## 详细介绍一下finalizer，并结合刚才提供的那个代码片段

ChatGPT

在 Kubernetes 中，Finalizer 是一种资源清理机制，用于处理资源被删除时需要进行的一些附加操作。在 Kubernetes API 中，资源的删除通常是异步进行的。当 API 收到一个删除资源的请求时，它并不会立即从存储中删除这个资源，而是会在资源的元数据中添加一个字段 deletionTimestamp，并且保留该资源，直到所有的 Finalizers 都被移除。这就允许我们在资源被彻底删除前执行一些自定义操作，比如清理子资源，或者更新外部系统的状态。

每个 Finalizer 都是一个字符串，通常表示了要执行的操作的类型。例如，Kubernetes 内置的 Finalizer kubernetes 表示的是删除资源时要删除其所有的依赖对象。自定义的 Finalizer 则可以在自定义资源的控制器中实现。

在你提供的代码片段中，一个名为 namespace.finalizer.tenancy.acme.com 的 Finalizer 被添加到了一个自定义资源 AcmeNamespace 的元数据中：

yaml

Copy code

apiVersion: tenancy.acme.com/v1alapha1

kind: AcmeNamespace

metadata:

name: team-x

finalizers:

- namespace.finalizer.tenancy.acme.com

spec:

namespaceName: app-y

adminUsername: sam

在这个例子中，当这个 AcmeNamespace 资源被删除时，Kubernetes API 服务器会先检查资源的 finalizers 字段，看到存在一个 Finalizer namespace.finalizer.tenancy.acme.com，然后将删除操作暂停，同时将 deletionTimestamp 字段设置为当前时间。

这个更改会触发 AcmeNamespace 控制器进行 reconciliation（调和）。控制器应该检查 deletionTimestamp 字段，如果它被设置了，那么控制器就知道这个资源正在被删除，可以执行相应的清理操作。完成后，控制器需要将 Finalizer namespace.finalizer.tenancy.acme.com 从 finalizers 列表中移除，然后更新资源。

当 API 服务器看到一个资源的 finalizers 列表为空时，它就会彻底删除这个资源。如果在清理操作完成前，API 服务器发现 finalizers 列表非空，那么它就会阻止删除操作。这就是 Finalizer 如何阻止资源被删除，直到所有清理工作完成的原理。

在一些情况下，这个机制是非常有用的，比如当一个自定义资源与集群中的其他资源有关系，或者与集群外的系统有关系时，需要在资源被删除前进行一些操作，例如清理子资源，或者更新外部系统的状态。