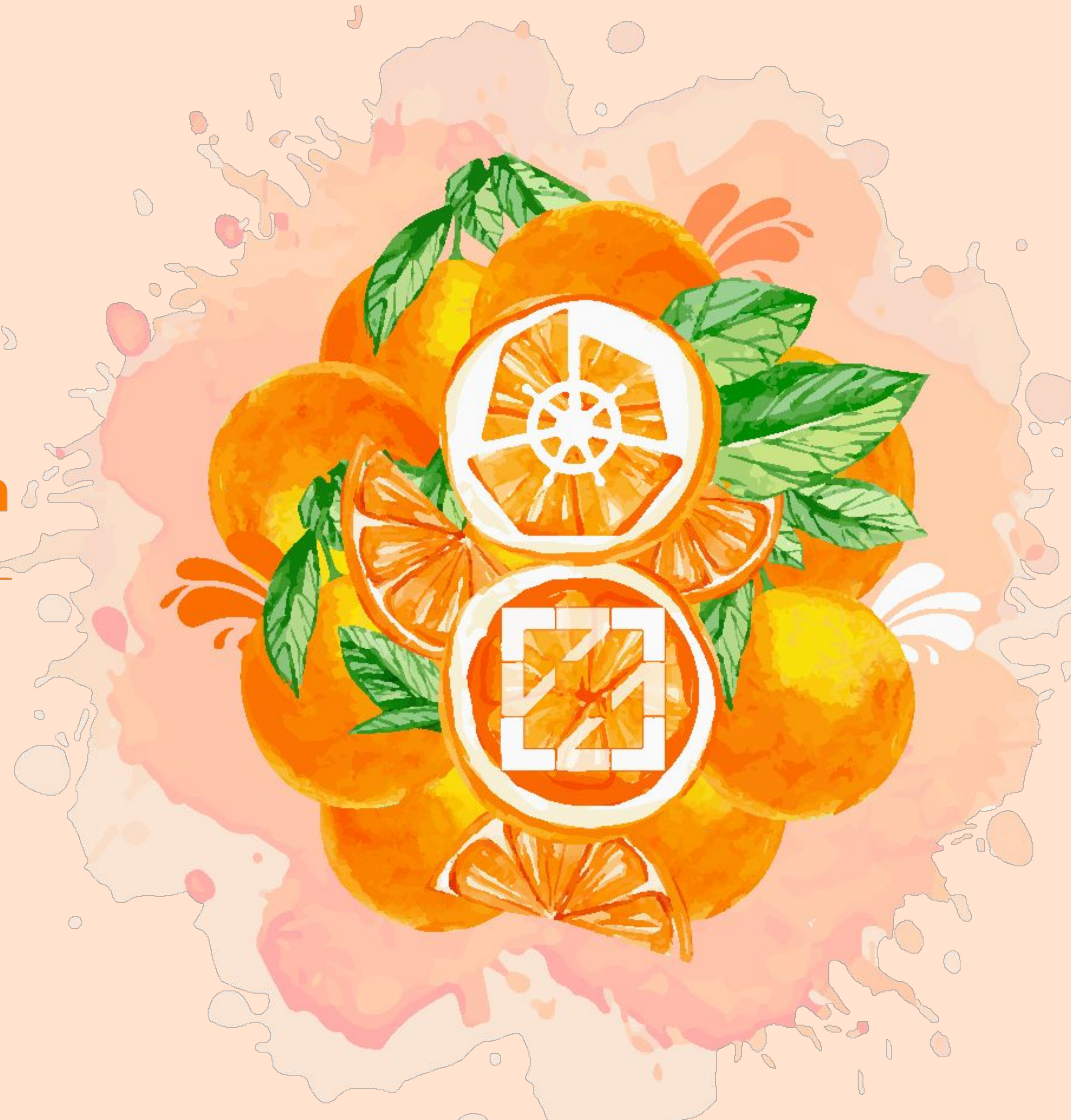
KubeCon | CloudNativeCon

Europe 2022

WELCOME TO VALENCIA



# Implementing Anti-patterns:

## Cross-namespace Resource Ownership

Tom Coufal, Red Hat

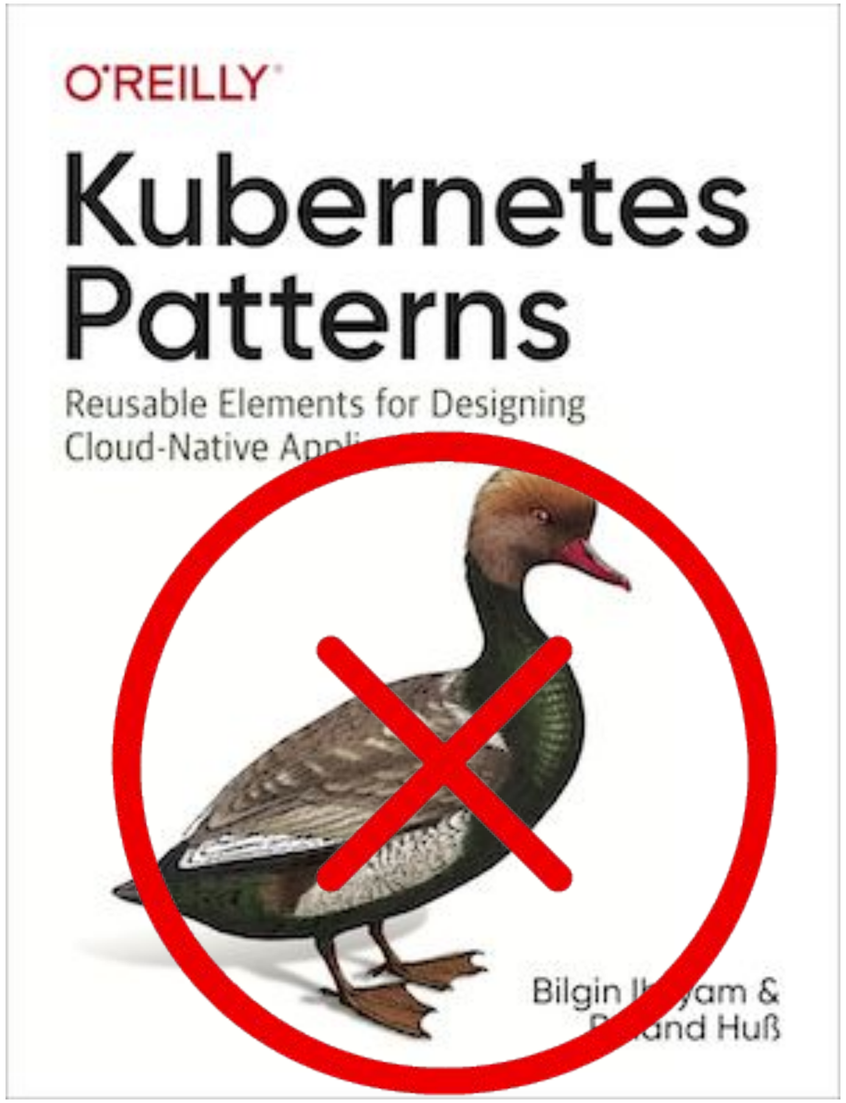
## Who?

**Tom Coufal**
Principal Software Engineer

Emerging Technologies @ Office of the CTO, Red Hat

github.com/tumido
linkedin.com/in/coufaltom

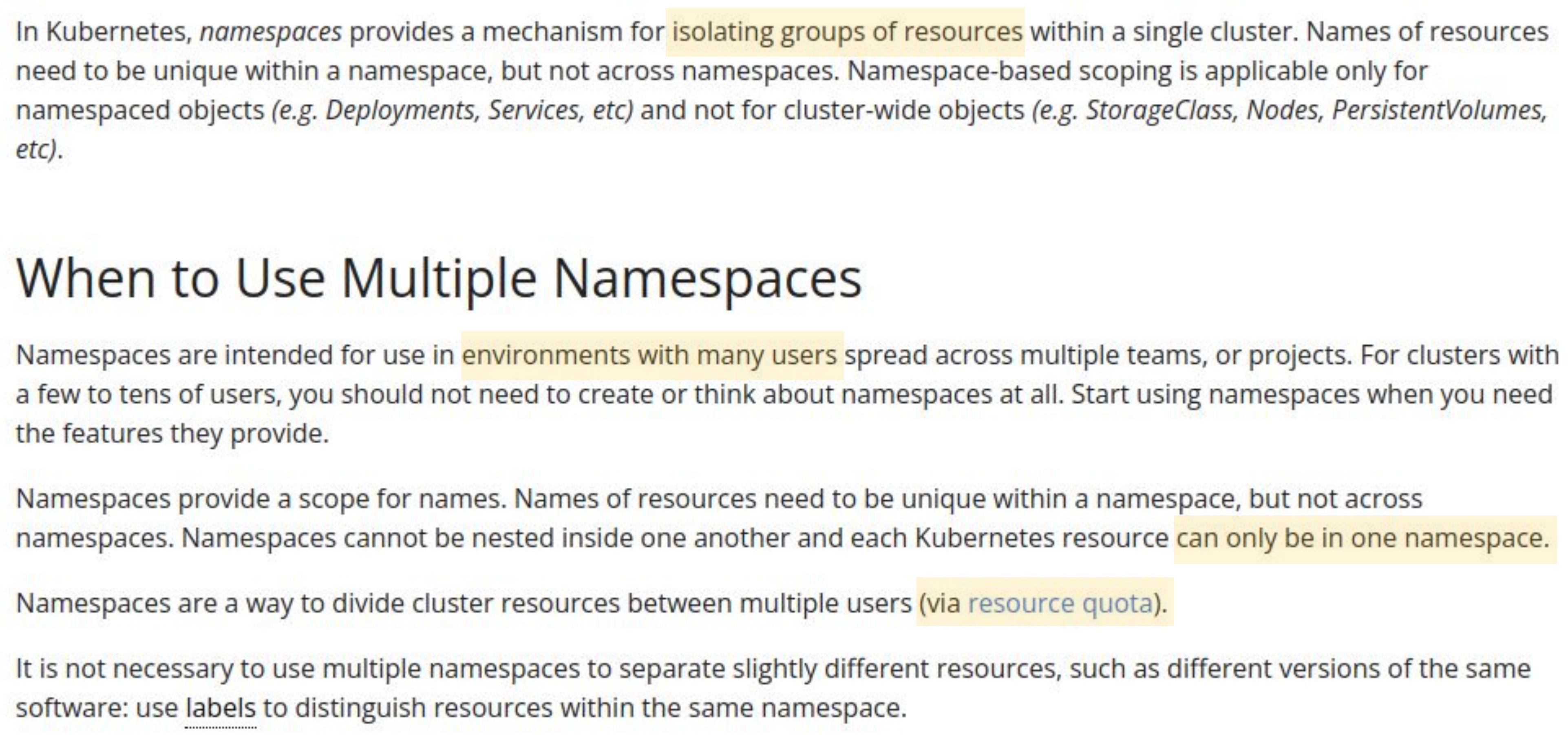
## What?


O'REILLY
Kubernetes Patterns
Reusable Elements for Designing
Cloud-Native Appli
Bilgin I yam &
and Huß

## How?

1. **Introduce concepts**
2. **Provoke with a use-case**
3. **Explore one solution**
4. **Implement it**
5. **Demo?**

## Namespaces



In Kubernetes, *namespaces* provides a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces. Namespace-based scoping is applicable only for namespaced objects *(e.g. Deployments, Services, etc)* and not for cluster-wide objects *(e.g. StorageClass, Nodes, PersistentVolumes, etc)*.

### When to Use Multiple Namespaces

Namespaces are intended for use in environments with many users spread across multiple teams, or projects. For clusters with a few to tens of users, you should not need to create or think about namespaces at all. Start using namespaces when you need the features they provide.

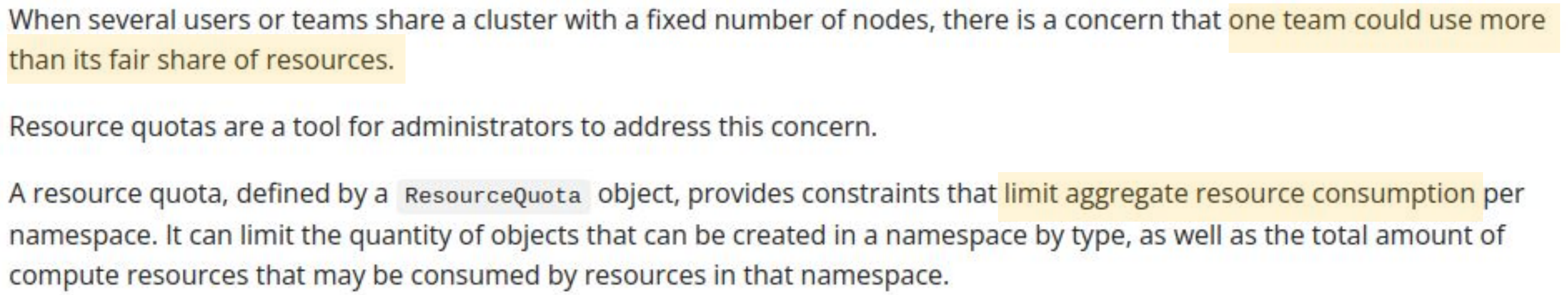
Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces. Namespaces cannot be nested inside one another and each Kubernetes resource can only be in one namespace.

Namespaces are a way to divide cluster resources between multiple users (via resource quota).

It is not necessary to use multiple namespaces to separate slightly different resources, such as different versions of the same software: use labels to distinguish resources within the same namespace.

https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

## Resource Quota



When several users or teams share a cluster with a fixed number of nodes, there is a concern that one team could use more than its fair share of resources.

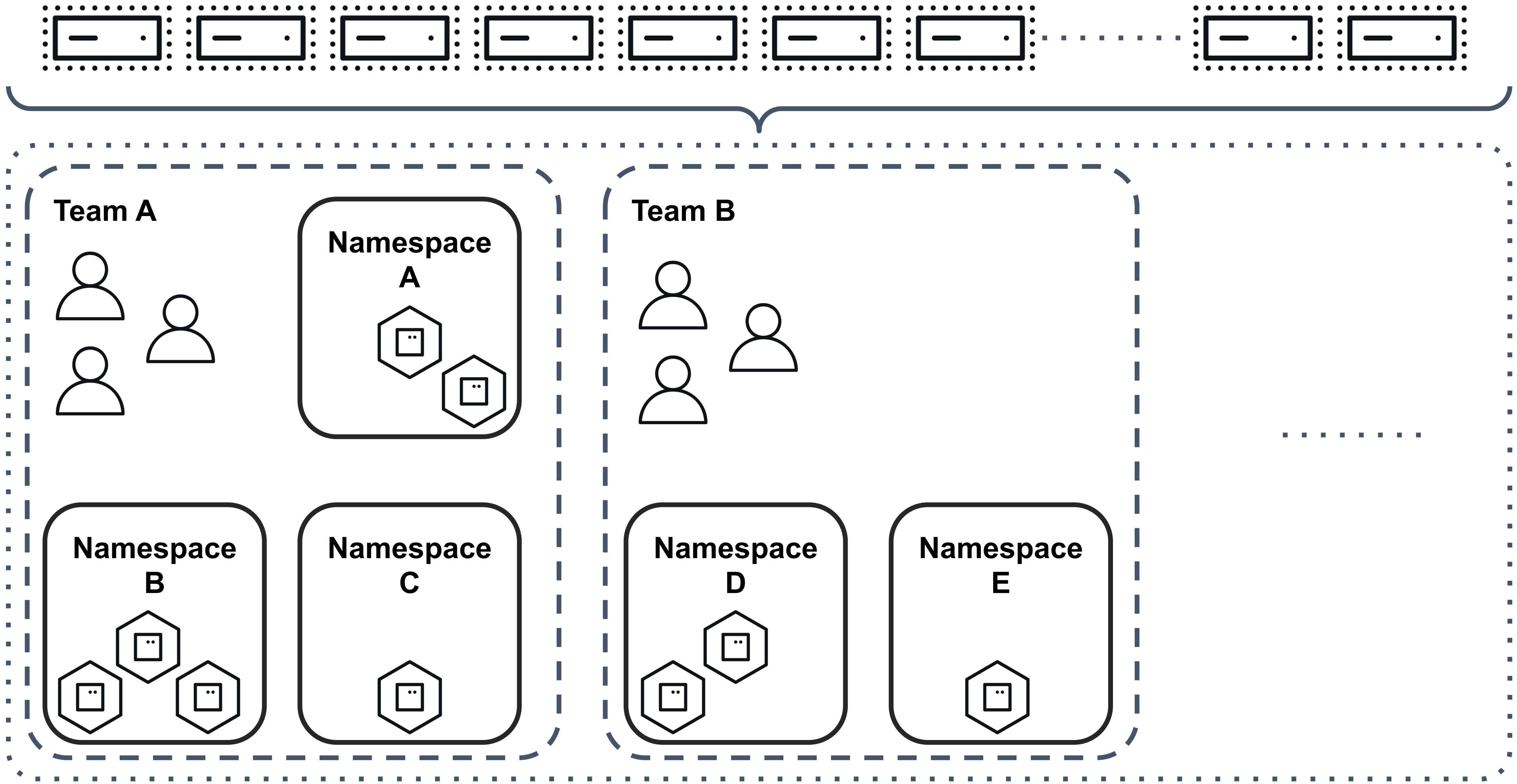
Resource quotas are a tool for administrators to address this concern.

A resource quota, defined by a `ResourceQuota` object, provides constraints that limit aggregate resource consumption per namespace. It can limit the quantity of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed by resources in that namespace.

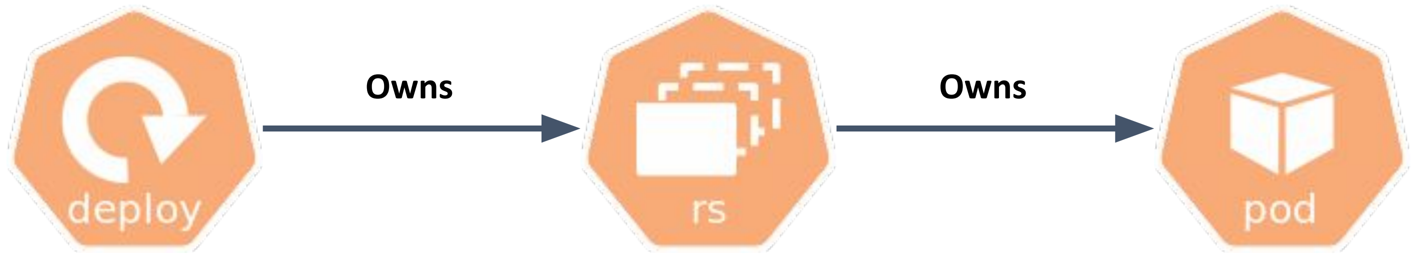
https://kubernetes.io/docs/concepts/policy/resource-quotas/

# Multitenancy

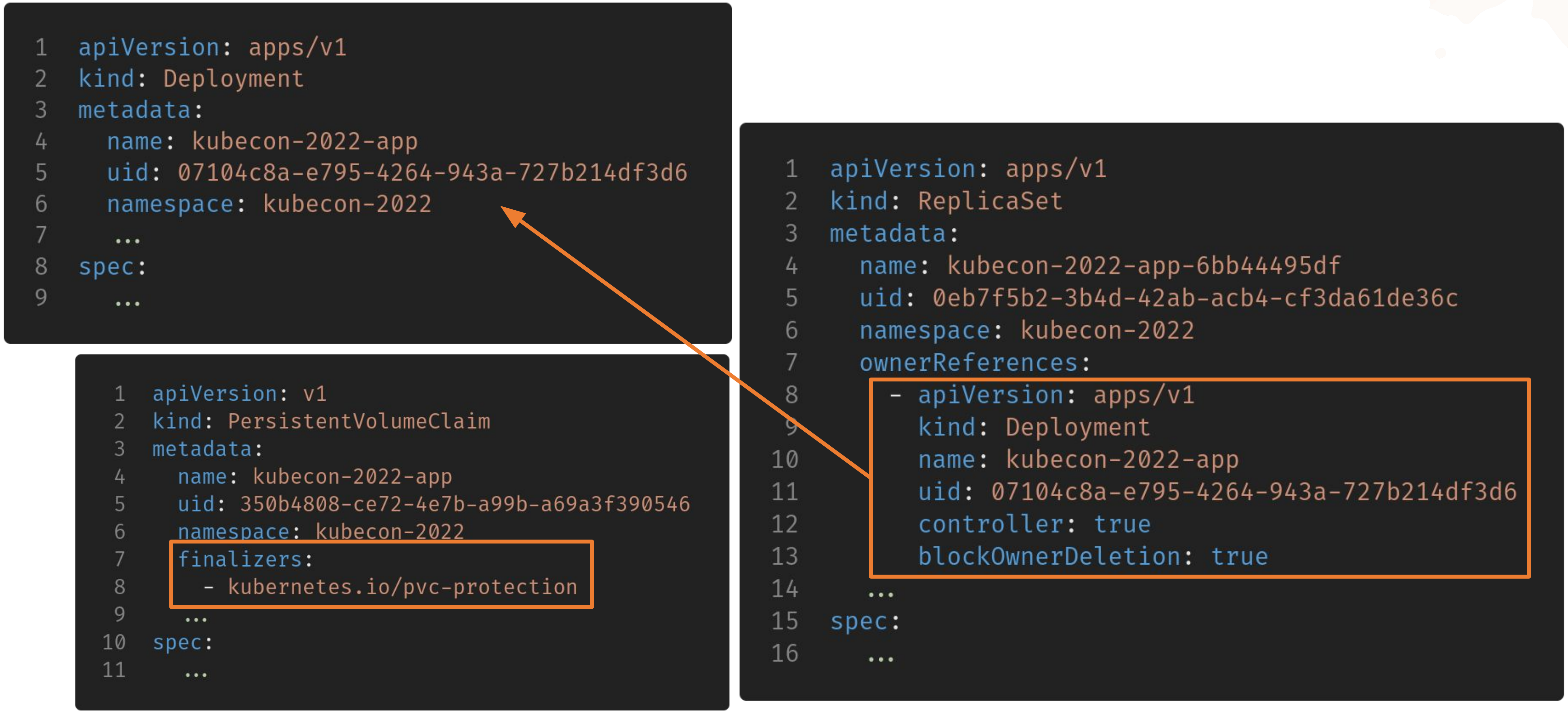
KubeCon | CloudNativeCon Europe 2022



## Owners and dependants



## Owners and dependants



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubecon-2022-app
  uid: 07104c8a-e795-4264-943a-727b214df3d6
  namespace: kubecon-2022
  ...
spec:
  ...
```

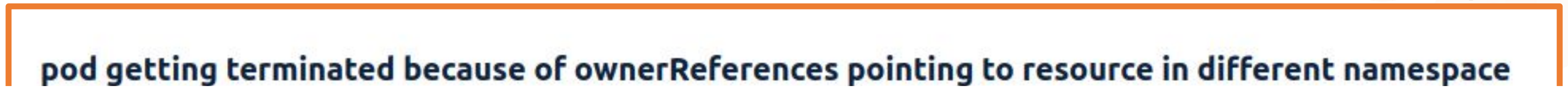
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: kubecon-2022-app
  uid: 350b4808-ce72-4e7b-a99b-a69a3f390546
  namespace: kubecon-2022
  finalizers:
    - kubernetes.io/pvc-protection
  ...
spec:
  ...
```

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: kubecon-2022-app-6bb44495df
  uid: 0eb7f5b2-3b4d-42ab-acb4-cf3da61de36c
  namespace: kubecon-2022
  ownerReferences:
    - apiVersion: apps/v1
      kind: Deployment
      name: kubecon-2022-app
      uid: 07104c8a-e795-4264-943a-727b214df3d6
      controller: true
      blockOwnerDeletion: true
  ...
spec:
  ...
```

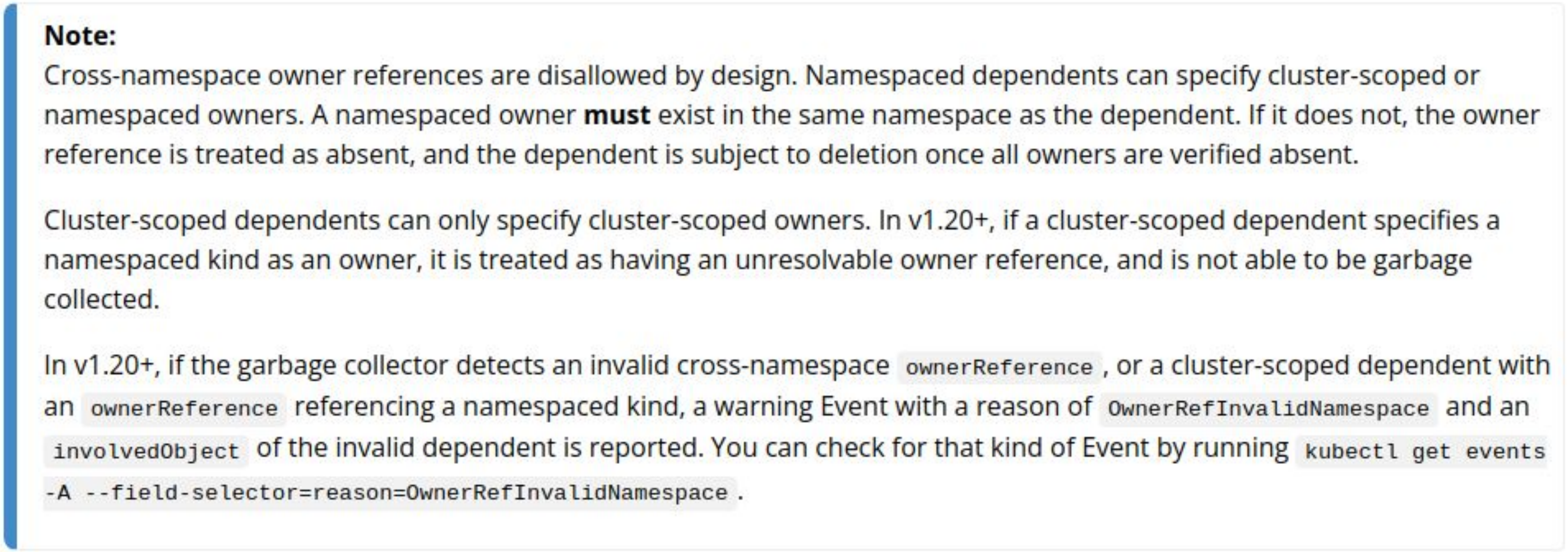
## Owners and dependants

| | | Owner | |
|---|---|---|---|
| | | Cluster-scoped | Namespaced |
| Dependant | Namespaced | ☑ | ☑ ? |
| | Cluster-scoped | ☑ | ⊗ |

## Owners and dependants



pod getting terminated because of ownerReferences pointing to resource in different namespace

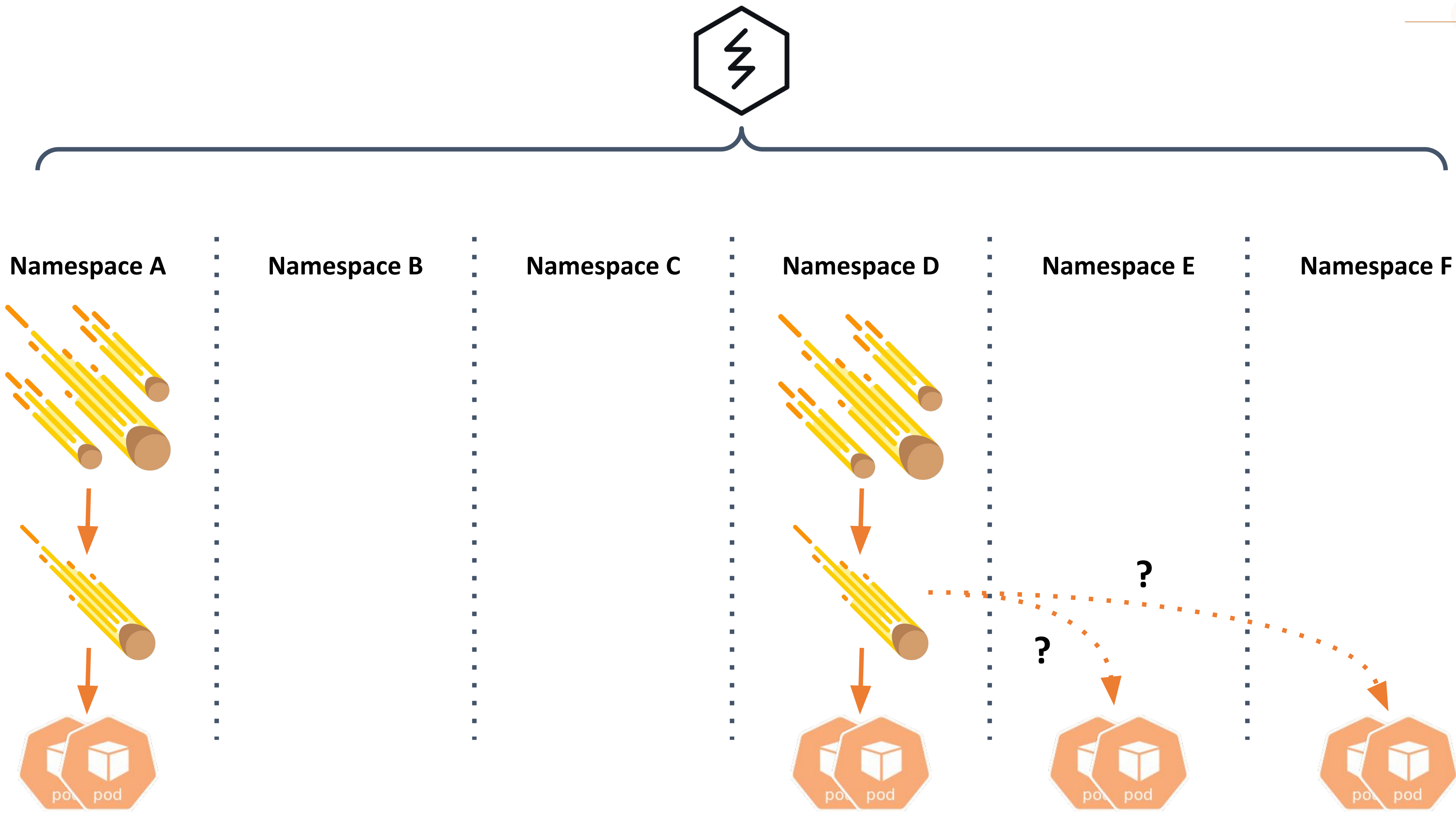


**Note:**
Cross-namespace owner references are disallowed by design. Namespaced dependents can specify cluster-scoped or namespaced owners. A namespaced owner **must** exist in the same namespace as the dependent. If it does not, the owner reference is treated as absent, and the dependent is subject to deletion once all owners are verified absent.

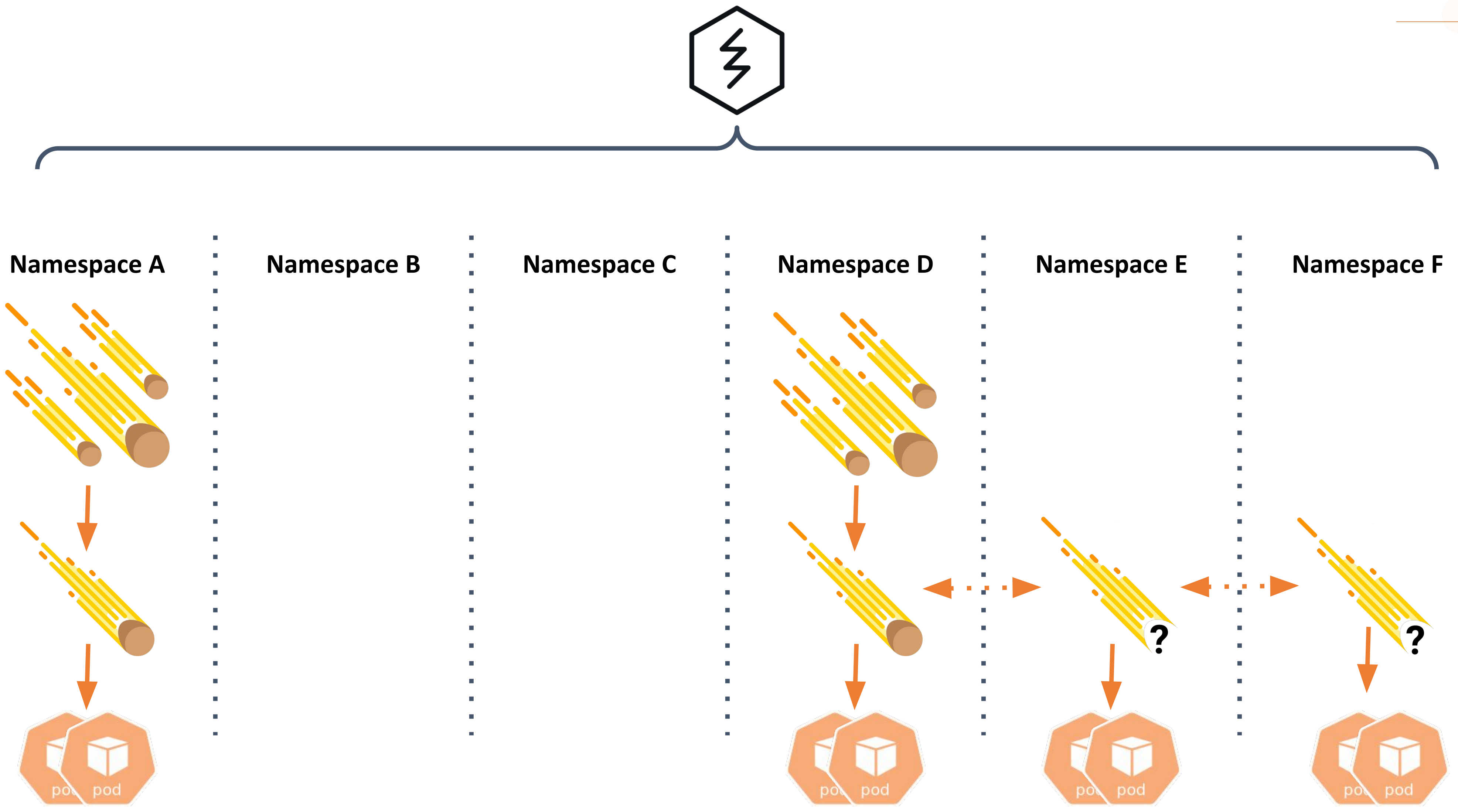
Cluster-scoped dependents can only specify cluster-scoped owners. In v1.20+, if a cluster-scoped dependent specifies a namespaced kind as an owner, it is treated as having an unresolvable owner reference, and is not able to be garbage collected.

In v1.20+, if the garbage collector detects an invalid cross-namespace `ownerReference` , or a cluster-scoped dependent with an `ownerReference` referencing a namespaced kind, a warning Event with a reason of `OwnerRefInvalidNamespace` and an `involvedObject` of the invalid dependent is reported. You can check for that kind of Event by running `kubectl get events -A --field-selector=reason=OwnerRefInvalidNamespace` .

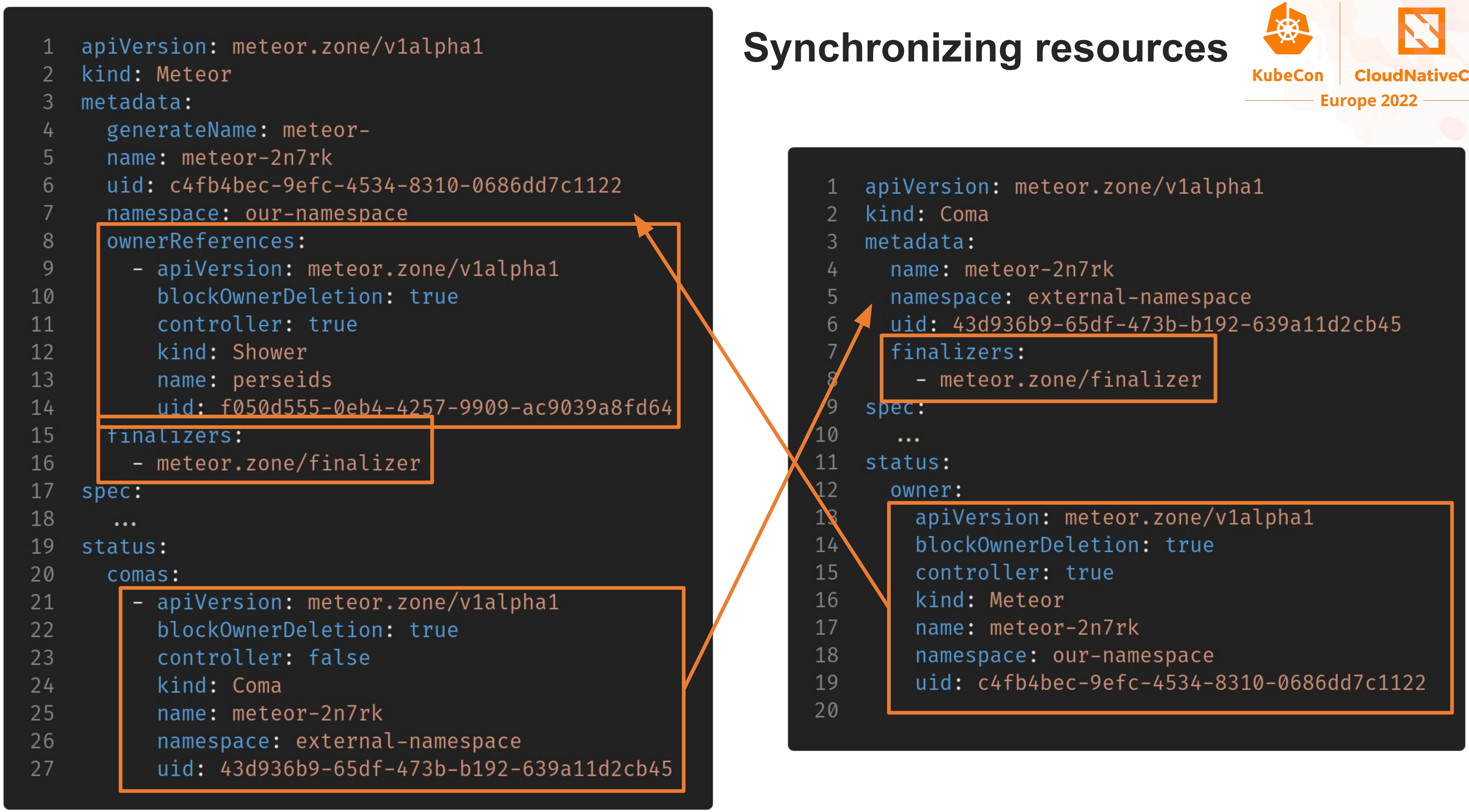
# Problem statement



# Problem statement



# Synchronizing resources



```yaml
apiVersion: meteor.zone/v1alpha1
kind: Meteor
metadata:
  generateName: meteor-
  name: meteor-2n7rk
  uid: c4fb4bec-9efc-4534-8310-0686dd7c1122
  namespace: our-namespace
  ownerReferences:
    - apiVersion: meteor.zone/v1alpha1
      blockOwnerDeletion: true
      controller: true
      kind: Shower
      name: perseids
      uid: f050d555-0eb4-4257-9909-ac9039a8fd64
  finalizers:
    - meteor.zone/finalizer
spec:
  ...
status:
  comas:
    - apiVersion: meteor.zone/v1alpha1
      blockOwnerDeletion: true
      controller: false
      kind: Coma
      name: meteor-2n7rk
      namespace: external-namespace
      uid: 43d936b9-65df-473b-b192-639a11d2cb45
```

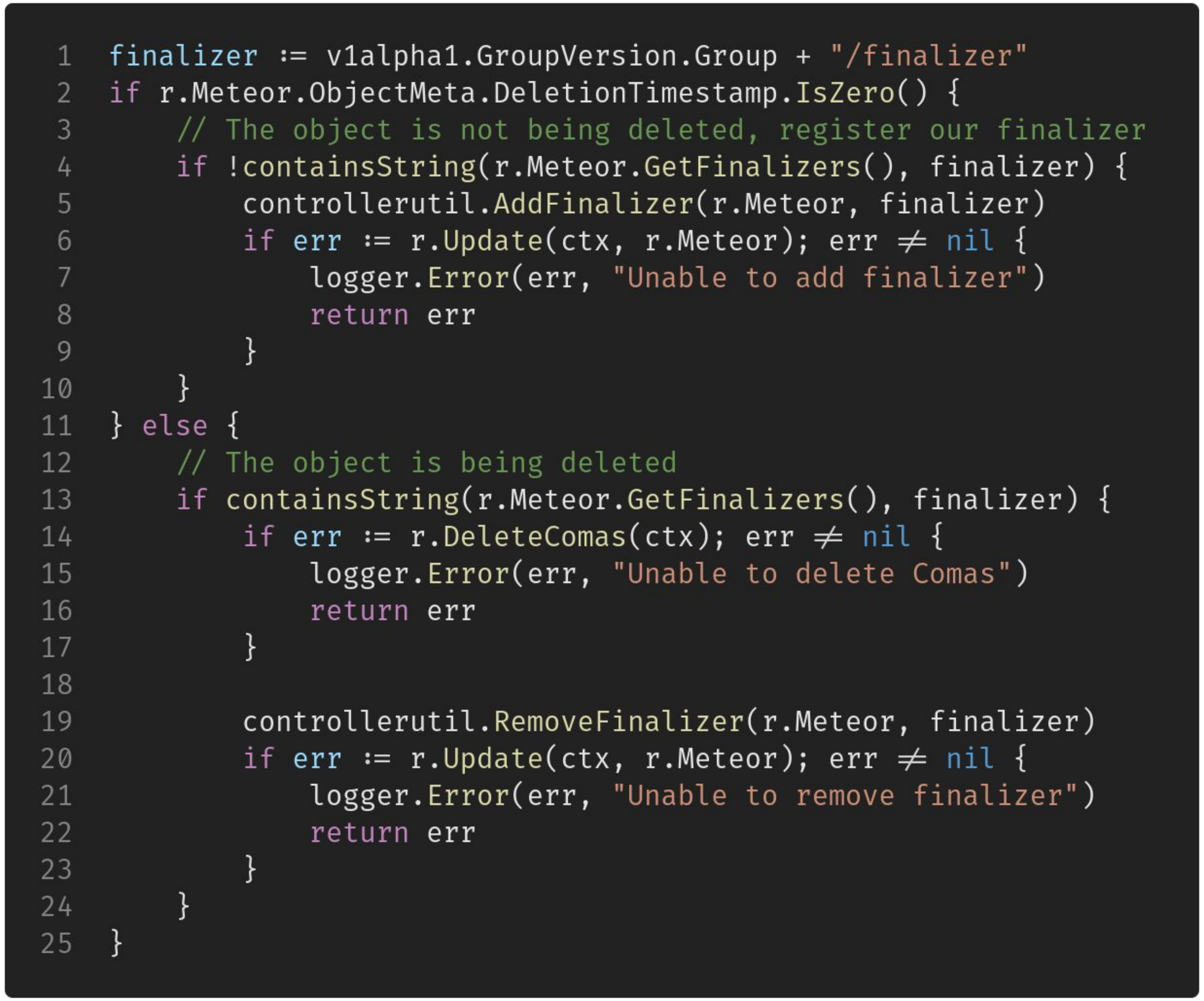
```yaml
apiVersion: meteor.zone/v1alpha1
kind: Coma
metadata:
  name: meteor-2n7rk
  namespace: external-namespace
  uid: 43d936b9-65df-473b-b192-639a11d2cb45
  finalizers:
    - meteor.zone/finalizer
spec:
  ...
status:
  owner:
    apiVersion: meteor.zone/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: Meteor
    name: meteor-2n7rk
    namespace: our-namespace
    uid: c4fb4bec-9efc-4534-8310-0686dd7c1122

```

## Implement it!

kubebuilder

OPERATOR
SDK

## Synchronizing owner resources



```go
finalizer := v1alpha1.GroupVersion.Group + "/finalizer"
if r.Meteor.ObjectMeta.DeletionTimestamp.IsZero() {
    // The object is not being deleted, register our finalizer
    if !containsString(r.Meteor.GetFinalizers(), finalizer) {
        controllerutil.AddFinalizer(r.Meteor, finalizer)
        if err := r.Update(ctx, r.Meteor); err ≠ nil {
            logger.Error(err, "Unable to add finalizer")
            return err
        }
    }
} else {
    // The object is being deleted
    if containsString(r.Meteor.GetFinalizers(), finalizer) {
        if err := r.DeleteComas(ctx); err ≠ nil {
            logger.Error(err, "Unable to delete Comas")
            return err
        }

        controllerutil.RemoveFinalizer(r.Meteor, finalizer)
        if err := r.Update(ctx, r.Meteor); err ≠ nil {
            logger.Error(err, "Unable to remove finalizer")
            return err
        }
    }
}
```

## Synchronizing owner resources

```go
func (r *MeteorReconciler) DeleteComas(ctx context.Context) error {
    logger := log.FromContext(ctx)
    for _, coma := range r.Meteor.Status.Comas {
        comaMeta := &v1alpha1.Coma{
            ObjectMeta: metav1.ObjectMeta{Name: coma.Name, Namespace: coma.Namespace},
        }
        logger.WithValues("coma", comaMeta).Info("Deleting coma")
        if err := r.Delete(ctx, comaMeta); err ≠ nil {
            logger.WithValues("coma", comaMeta).Error(err, "Failed to delete coma")
            return err
        }
    }
    return nil
}
```

## Synchronizing owner resources

KubeCon CloudNativeCon Europe 2022

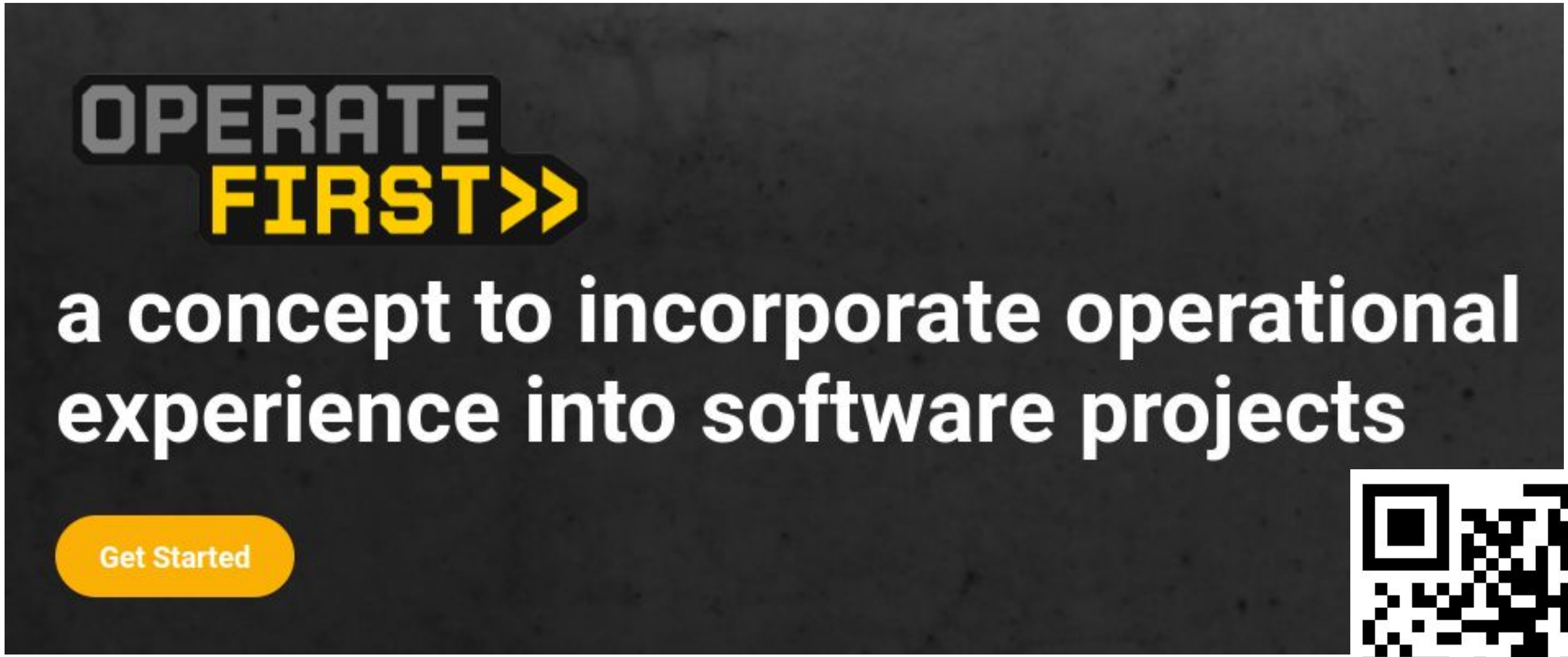
```
for _, externalService := range r.Shower.Spec.ExternalServices {
```

```
coma := &v1alpha1.Coma{}
namespacedName := types.NamespacedName{
    Name: r.Meteor.GetName(),
    Namespace: externalService.Namespace,
}

if err := r.Get(ctx, namespacedName, coma); err ≠ nil {
    if errors.IsNotFound(err) {
        coma = &v1alpha1.Coma{
            ObjectMeta: metav1.ObjectMeta{
                Name:      r.Meteor.GetName(),
                Namespace: externalService.Namespace,
            },
        }
        if err := r.Create(ctx, coma); err ≠ nil {
            return err
        }
    }
}
```

```
ref := v1alpha1.NamespacedOwnerReference{
    OwnerReference: *metav1.NewControllerRef( … ),
    Namespace:      externalService.Namespace,
}
ref.Controller = pointer.BoolPtr(false)
if !containsComa(r.Meteor.Status.Comas, ref) {
    r.Meteor.Status.Comas = append(r.Meteor.Status.Comas, ref)
}

coma.Status.Owner = r.Meteor.GetReference(true)

if err := r.Status().Update(ctx, coma); err ≠ nil {
    // …
}
```

```
}
```

## Deploy and run

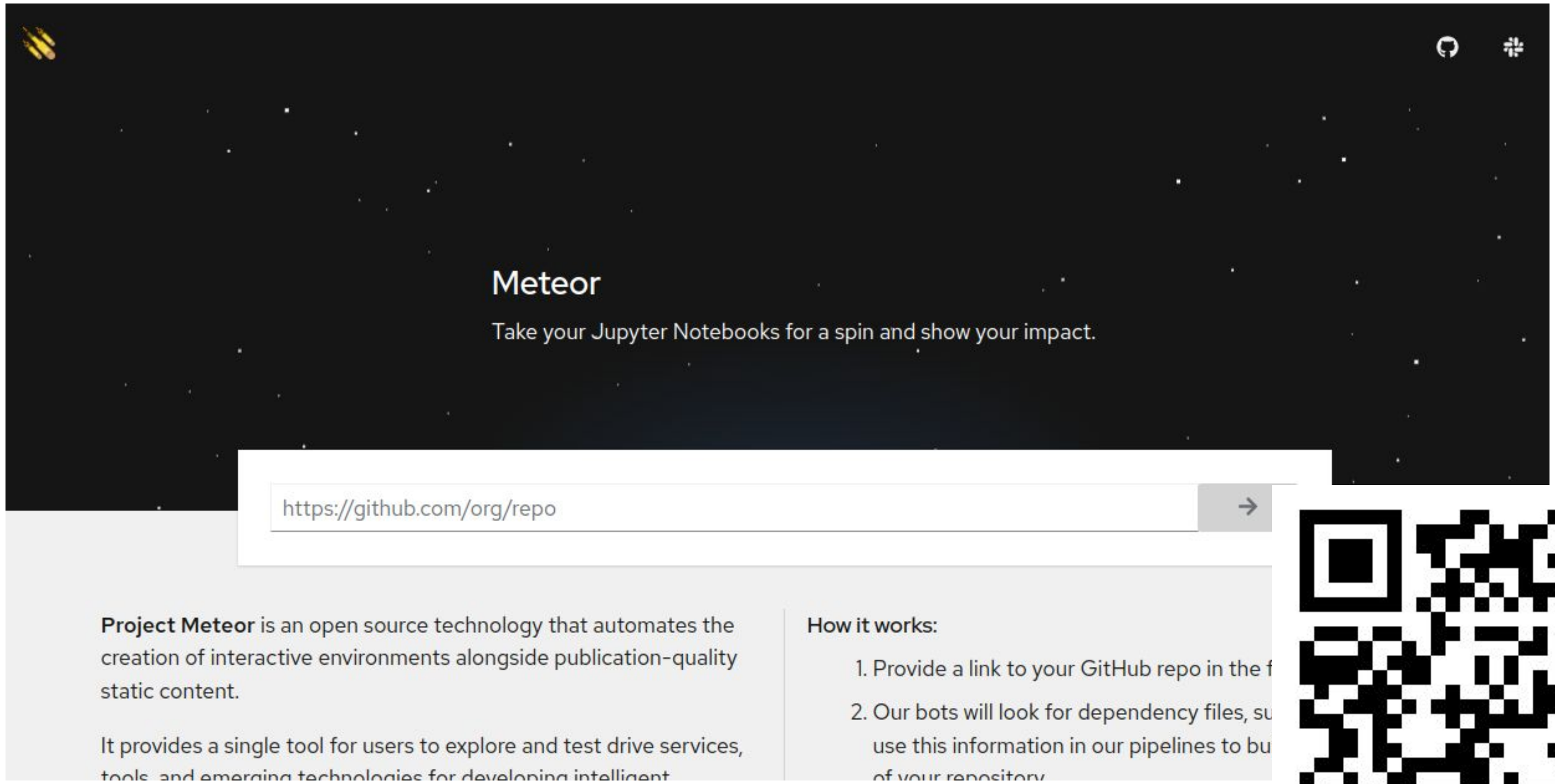


OPERATE FIRST>>

a concept to incorporate operational experience into software projects

## Demo

KubeCon CloudNativeCon Europe 2022



Meteor

Take your Jupyter Notebooks for a spin and show your impact.

https://github.com/org/repo

**Project Meteor** is an open source technology that automates the creation of interactive environments alongside publication-quality static content.

It provides a single tool for users to explore and test drive services, tools, and emerging technologies for developing intelligent

**How it works:**

1. Provide a link to your GitHub repo in the f
2. Our bots will look for dependency files, su use this information in our pipelines to bu of your repository

https://shower.meteor.zone

## Try writing controllers, extend Kubernetes!

https://github.com/
AICoE/meteor-operator/

Operator SDK

Kubebuilder SDK
https://book.kubebuilder.io/