KUBERNETES
COMMUNITY DAYS CHINA 2022

Karmada Resource Interpreter Webhook 解析

深度支持自定义资源（CRD）

徐信钊@QingCloud

Karmada Resource Interpreter Webhook 解析

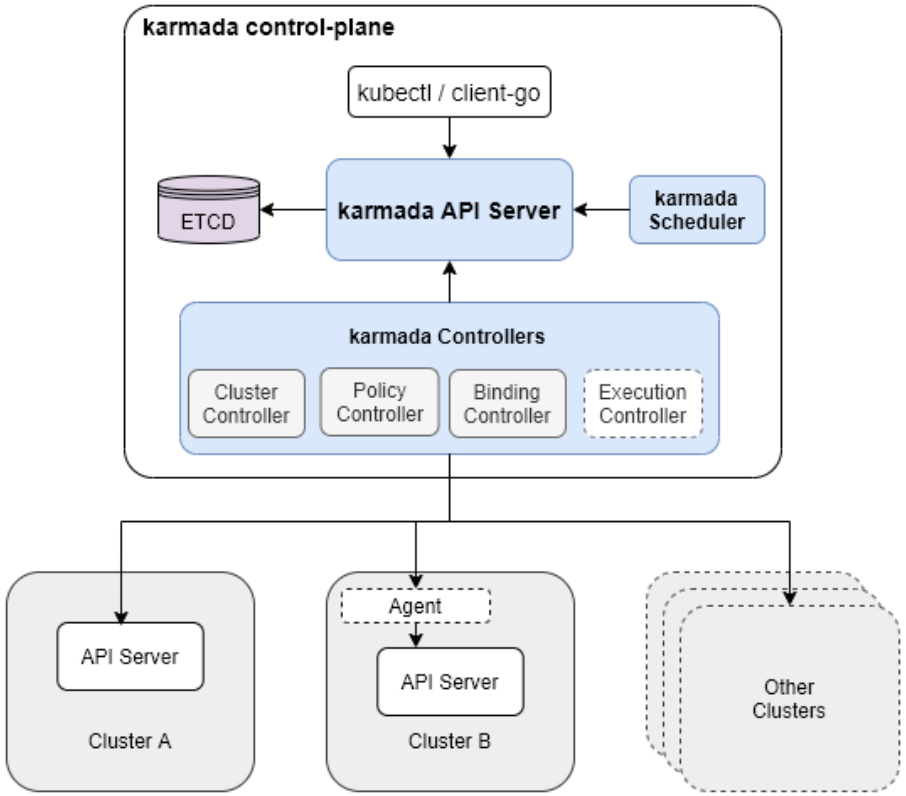
KUBERNETES COMMUNITY DAYS CHINA 2022

目录

Karmada Resource Interpreter Webhook 解析

Karmada 简介

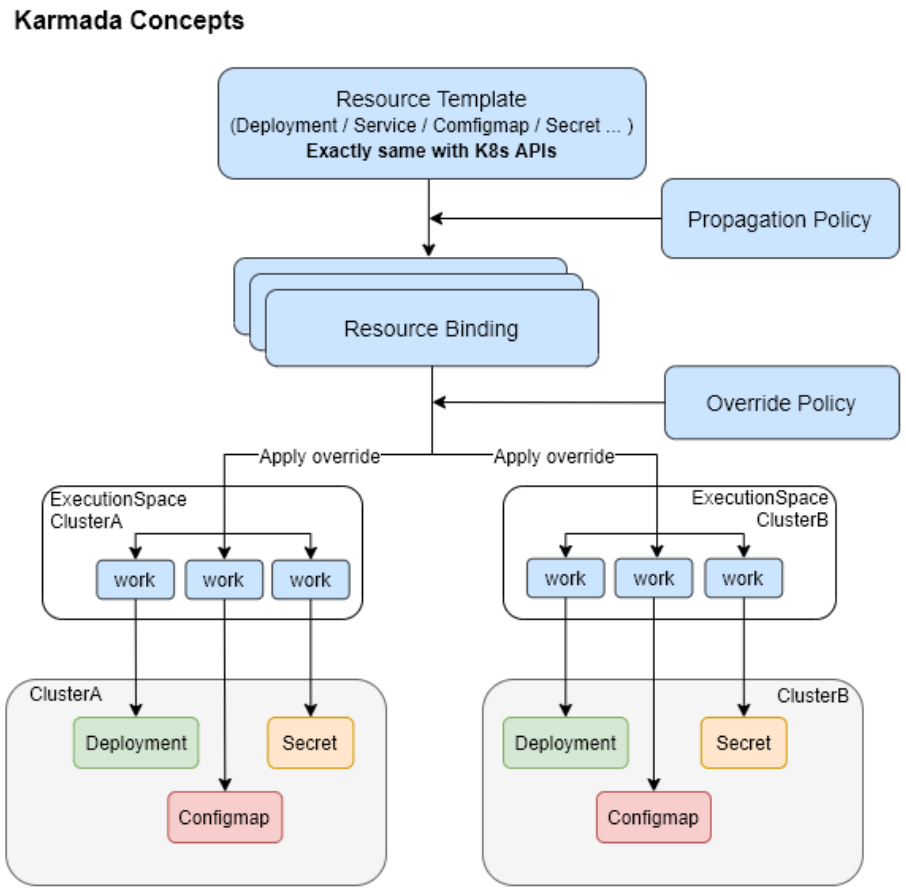
Karmada 是一个开源的多云容器编排项目，这个项目是 Kubernetes Federation v1 和 v2 的延续，一些基本概念继承自这两个版本。



- Karmada API Server：本质就是一个普通的 K8s API Server，绑定了一个单独的 etcd 来存储那些要被联邦托管的资源
- Karmada Controller Manager：多个 controller 的集合，监听 Karmada API Server 中的对象并与成员集群 API server 进行通信
- Karmada Scheduler：提供高级的多集群调度策略

Karmada Resource Interpreter Webhook 解析

Karmada 简介



一些基本概念：

- 资源模板（Resource Template）：Karmada 使用 K8s 原生 API 定义作为资源模板，便于快速对接 K8s 生态工具链
- 分发策略（Propagaion Policy）：Karmada 提供独立的策略 API，用来配置资源分发策略
- 差异化策略（Override Policy）：Karmada 提供独立的差异化 API，用来配置与集群相关的差异化配置，比如配置不同集群使用不同的镜像

Karmada Resource Interpreter Webhook 解析

一个例子：创建一个 nginx 应用

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
```

```yaml
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: nginx-propagation
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: nginx
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
```

创建一个副本数为 2 的 nginx deployment
并将其直接分发到 member1 和 member2 集群

Karmada Resource Interpreter Webhook 解析

一个例子：创建一个 nginx 应用

```
$ kubectl create -f samples/nginx
deployment.apps/nginx created
propagationpolicy.policy.karmada.io/nginx-propagation created

$ kubectl get rb
NAME               SCHEDULED   FULLYAPPLIED   AGE
nginx-deployment   True        True           16s

$ kubectl get work -A
NAMESPACE             NAME               APPLIED   AGE
karmada-es-member1   nginx-687f7fb96f   True      22s
karmada-es-member2   nginx-687f7fb96f   True      22s

$ kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   4/2     4            4           35s
```

member1 和 member2 集群分别有一个副本数为 2 的 nginx deployment，所以该模板资源一共存在 4 个 Pod

Karmada Resource Interpreter Webhook 解析

创建 nginx 应用引出的问题

上面的例子非常简单，直接在 member 集群根据模板原封不动创建 deployment 就行了，但是大家知道 Karmada 是支持一些更高级的副本数调度策略的，比如下面这个例子：

```yaml
replicaScheduling:
  replicaDivisionPreference: Weighted
  replicaSchedulingType: Divided
  weightPreference:
    staticWeightList:
      - targetCluster:
          clusterNames:
            - member1
        weight: 1
      - targetCluster:
          clusterNames:
            - member2
        weight: 1
```

应用了该规则之后，会涉及到针对每个集群上资源副本数的动态调整，之后 Karmada 在 member 集群创建 deployment 的时候就需要增加一个修改副本数的步骤。

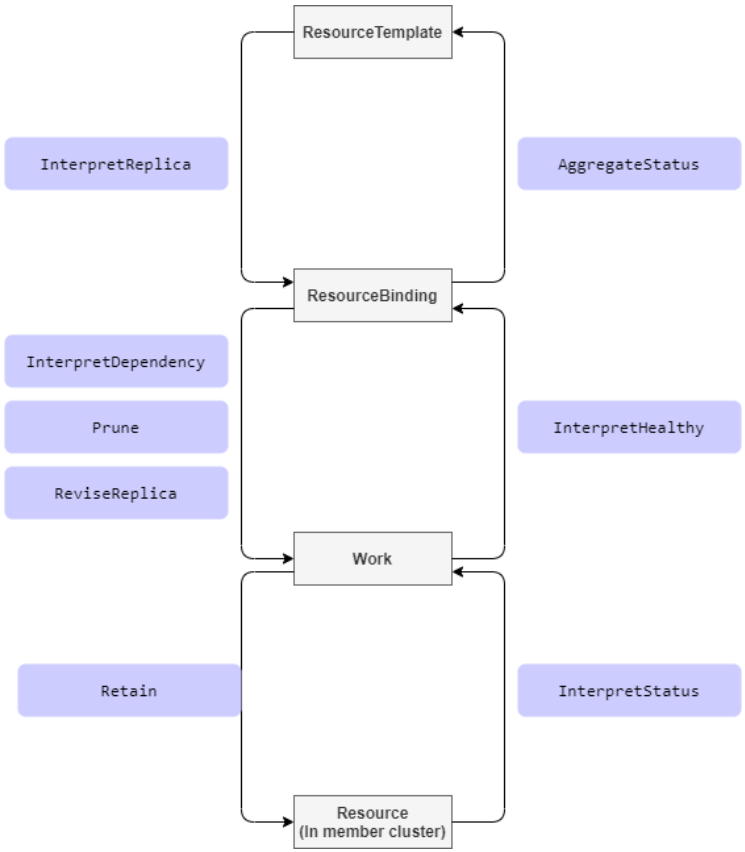
创建 nginx 应用引出的问题

针对 deployment 这类 K8s 核心资源，因为其结构是确定的，我们可以直接编写修改其副本数的代码，但是如果我有一个功能类似 deployment 的 CRD 呢？我也需要副本数调度，Karmada 能正确地修改它的副本数吗？

Karmada Resource Interpreter Webhook 解析

Resource Interpreter Webhook

为了解决上面提到的问题，Karmada 引入了 Resource Interpreter Webhook，通过干预从 ResourceTemplate 到 ResourceBinding 到 Work 到 Resource 的这几个阶段来实现完整的自定义资源分发能力。



从一个阶段到另一个都会经过我们预定义的一个或多个接口，我们会在这些步骤中实现修改副本数等操作。

用户需要增加一个单独的实现了对应接口的 webhook server，Karmada 会在执行到相应步骤时去调用该 server 来完成操作。

Karmada Resource Interpreter Webhook 解析

KUBERNETES COMMUNITY DAYS CHINA 2022

Resource Interpreter Webhook

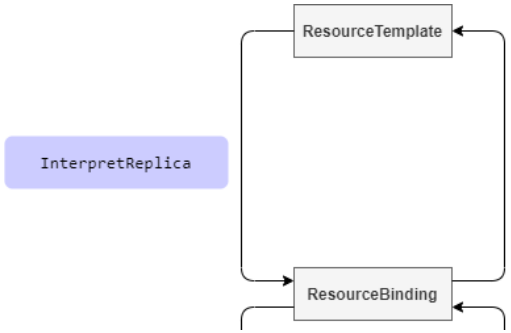
接下来都以下面这个 Workload CRD 作为示例

```go
// Workload is a simple Deployment.
type Workload struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    // Spec represents the specification of the desired behavior.
    // +required
    Spec WorkloadSpec `json:"spec"`

    // Status represents most recently observed status of the Workload.
    // +optional
    Status WorkloadStatus `json:"status,omitempty"`
}

// WorkloadSpec is the specification of the desired behavior of the Workload.
type WorkloadSpec struct {
    // Number of desired pods. This is a pointer to distinguish between explicit
    // zero and not specified. Defaults to 1.
    // +optional
    Replicas *int32 `json:"replicas,omitempty"`

    // Template describes the pods that will be created.
    Template corev1.PodTemplateSpec `json:"template" protobuf:"bytes,3,opt,name=template"`

    // Paused indicates that the deployment is paused.
    // Note: both user and controllers might set this field.
    // +optional
    Paused bool `json:"paused,omitempty"`
}

// WorkloadStatus represents most recently observed status of the Workload.
type WorkloadStatus struct {
    // ReadyReplicas represents the total number of ready pods targeted by this Workload.
    // +optional
    ReadyReplicas int32 `json:"readyReplicas,omitempty"`
}
```

Karmada Resource Interpreter Webhook 解析

InterpretReplica

针对有 replica 功能的资源对象，比如类似 Deployment 的自定义资源，实现该接口来告诉 Karmada 对应资源的副本数




```yaml
apiVersion: workload.example.io/v1alpha1
kind: Workload
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx
          name: nginx
```

Karmada Resource Interpreter Webhook 解析

InterpretReplica

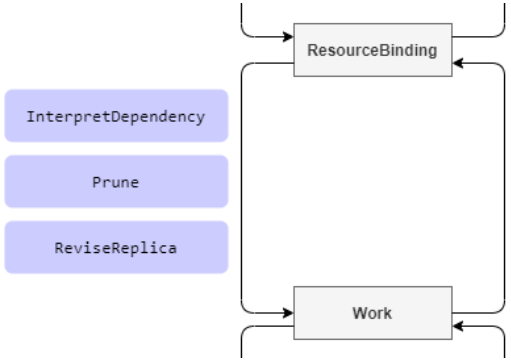
```go
func (e *workloadInterpreter) responseWithExploreReplica(workload *workloadv1alpha1.Workload) interpreter.Response {
    res := interpreter.Succeeded("")
    res.Replicas = workload.Spec.Replicas
    return res
}
```

直接返回该对象的副本数即可

Karmada Resource Interpreter Webhook 解析

ReviseReplica

针对有 replica 功能的资源对象，需要按照 Karmada 发送的 request 来修改对象的副本数



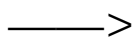

```go
func (e *workloadInterpreter) responseWithExploreReviseReplica(workload *workloadv1alpha1.Workload, req interpreter.Request) interpreter.Response {
    wantedWorkload := workload.DeepCopy()
    wantedWorkload.Spec.Replicas = req.DesiredReplicas
    marshaledBytes, err := json.Marshal(wantedWorkload)
    if err != nil {
        return interpreter.Errored(http.StatusInternalServerError, err)
    }
    return interpreter.PatchResponseFromRaw(req.Object.Raw, marshaledBytes)
}
```

Karmada Resource Interpreter Webhook 解析

Workload 实现副本数调度

结合 InterpretReplica 和 ReviseReplica hook 我们就能解决最开始提到的问题，为一个自定义资源实现副本数调度。

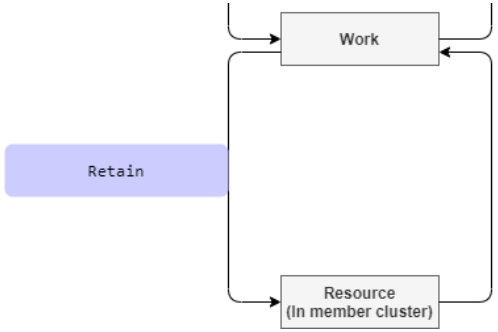
```yaml
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: nginx-workload-propagation
spec:
  resourceSelectors:
    - apiVersion: workload.example.io/v1alpha1
      kind: Workload
      name: nginx
  placement:
    clusterAffinity:
      clusterNames:
        - member1
        - member2
    replicaScheduling:
      replicaDivisionPreference: Weighted
      replicaSchedulingType: Divided
      weightPreference:
        staticWeightList:
          - targetCluster:
              clusterNames:
                - member1
            weight: 2
          - targetCluster:
              clusterNames:
                - member2
            weight: 1
```



```yaml
# karmada apiserver
$ kubectl get workload nginx -o yaml
apiVersion: workload.example.io/v1alpha1
kind: Workload
metadata:
  ...
spec:
  replicas: 3
  ...


# member1
$ kubectl get workload nginx -o yaml
apiVersion: workload.example.io/v1alpha1
kind: Workload
metadata:
  ...
spec:
  replicas: 2
  ...


# member2
apiVersion: workload.example.io/v1alpha1
kind: Workload
metadata:
  ...
spec:
  replicas: 1
  ...
```

Karmada Resource Interpreter Webhook 解析

Retain



针对 spec 内容会在 member 集群单独更新的情况，可以通过该 hook 告知 Karmada 保留某些字段的内容

```yaml
apiVersion: workload.example.io/v1alpha1
kind: Workload
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  paused: false
```

以 paused 为例，该字段的功能是暂停 workload，member 集群的 controller 会单独更新该字段，Retain hook 就是为了能更好地和 member 集群的 controller 协作，可以通过该 hook 来告知 Karmada 哪些字段是需要不用更新、需要保留的。

Karmada Resource Interpreter Webhook 解析

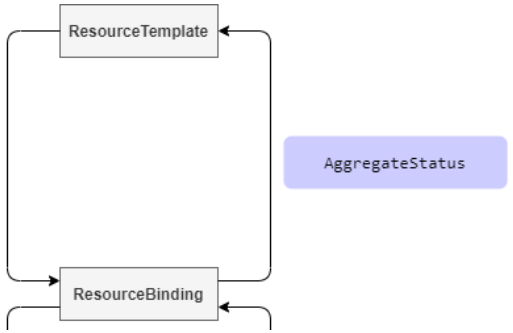
Retain

```go
func (e *workloadInterpreter) responseWithExploreRetaining(desiredWorkload *workloadv1alpha1.Workload, req interpreter.Request)
interpreter.Response {
    if req.ObservedObject == nil {
        err := fmt.Errorf("nil observedObject in exploreReview with operation type: %s", req.Operation)
        return interpreter.Errored(http.StatusBadRequest, err)
    }
    observerWorkload := &workloadv1alpha1.Workload{}
    err := e.decoder.DecodeRaw(*req.ObservedObject, observerWorkload)
    if err != nil {
        return interpreter.Errored(http.StatusBadRequest, err)
    }

    // Suppose we want to retain the `.spec.paused` field of the actual observed workload object in member cluster,
    // and prevent from being overwritten by karmada controller-plane.
    wantedWorkload := desiredWorkload.DeepCopy()
    wantedWorkload.Spec.Paused = observerWorkload.Spec.Paused
    marshaledBytes, err := json.Marshal(wantedWorkload)
    if err != nil {
        return interpreter.Errored(http.StatusInternalServerError, err)
    }
    return interpreter.PatchResponseFromRaw(req.Object.Raw, marshaledBytes)
}
```

核心代码只有一行，更新 wantedWorkload 的 Paused 字段为之前版本的内容

Karmada Resource Interpreter Webhook 解析

AggregateStatus



针对需要将 status 信息聚合到 Resource Template 的资源类型，可通过实现该接口来更新 Resource Template 的 status 信息

```
$ kubectl get rb nginx-deployment -o yaml

apiVersion: work.karmada.io/v1alpha2
kind: ResourceBinding
metadata:
  ...
spec:
  ...
status:
  aggregatedStatus:
  - applied: true
    clusterName: member1
    status:
      availableReplicas: 2
      observedGeneration: 1
      readyReplicas: 2
      replicas: 2
      updatedReplicas: 2
  - applied: true
    clusterName: member2
    status:
      availableReplicas: 2
      observedGeneration: 1
      readyReplicas: 2
      replicas: 2
      updatedReplicas: 2
```

Karmada 会将各个集群 Resouce 的状态信息统一收集到 ResourceBinding 中，AggregateStatus hook 需要做的事情就是将 ResourceBinding 中 status 信息更新到 Resource Template 中。

Karmada Resource Interpreter Webhook 解析

AggregateStatus

```go
func (e *workloadInterpreter) responseWithExploreAggregateStatus(workload *workloadv1alpha1.Workload, req interpreter.Request) interpreter.Response
{
    wantedWorkload := workload.DeepCopy()
    var readyReplicas int32
    for _, item := range req.AggregatedStatus {
        if item.Status == nil {
            continue
        }
        status := &workloadv1alpha1.WorkloadStatus{}
        if err := json.Unmarshal(item.Status.Raw, status); err != nil {
            return interpreter.Errored(http.StatusInternalServerError, err)
        }
        readyReplicas += status.ReadyReplicas
    }
    wantedWorkload.Status.ReadyReplicas = readyReplicas
    marshaledBytes, err := json.Marshal(wantedWorkload)
    if err != nil {
        return interpreter.Errored(http.StatusInternalServerError, err)
    }
    return interpreter.PatchResponseFromRaw(req.Object.Raw, marshaledBytes)
}
```

逻辑也非常简单，根据 ResourceBinding 中的 status 信息来计算（聚合）出该资源总的 status 信息

```
$ kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
nginx   4/2     4            4           23h
```

Karmada Resource Interpreter Webhook 解析

KUBERNETES COMMUNITY DAYS CHINA 2022

参考链接

- Resource Interpreter Webhook
- custom resource interpreter example

Karmada Resource Interpreter Webhook 解析

加入社区

我们会陆续实现其它的接口，欢迎大家使用和反馈，如果有必要，后续也可能增加更多的接口来满足更个性化的需求。

https://github.com/karmada-io/karmada

https://slack.cncf.io/ #karmada

容器魔方公众号

每日推送图文

社区最新动态

直播课程、技术干货

扫码添加小助手

发送“karmada”加群

社区专家入驻

技术问题随时答疑

KUBERNETES
COMMUNITY DAYS CHINA 2022

感谢观看

徐信钊@QingCloud