

96Kubernetes 系列（九十）Karmada 多云、多集群 Kubernetes 编排：第 2 部分

这是由两部分组成的系列中的第二部分。在第一部分中，我们讨论了多云、多集群 Kubernetes 的动机以及 Karmada 如何根据我们的要求将应用程序工作负载编排到各种集群中。我们还讨论了 Karmada 的概念、架构和功能。

在这一部分中，我们将使用 Karmada 进行一些实际操作，尝试将应用程序部署到多个集群中，并探索 Karmada 提供的各种选项/功能。

环境设置

我们将使用 Mac OS X 建立一个实验室环境。如果您愿意，也可以使用 Linux 操作系统。

在这里，我们在 Kubernetes 集群中安装 `karmada` 控制平面组件，称为 `host cluster`。然后我们将使用 `host cluster` 加入三个成员集群。`member1` 和 `member2` 集群将以 Push 模式加入主机，而 `member3` 将以 Pull 模式加入主机。

先决条件

- Go 版本 v1.18+
- kubectl 版本 v1.19+
- kind 版本 v0.14.0+

安装 Karmada 控制平面

1. 将此存储库克隆到您的计算机：

```
$ git clone https://github.com/karmada-io/karmada
```

2. 部署并运行 Karmada 控制平面：

```
$ cd karmada
$ hack/local-up-karmada.sh
```

该脚本将为我们执行以下任务：

- 启动 Kubernetes 集群来运行 Karmada 控制平面，也称为 `host cluster`。
- 基于当前代码库构建 Karmada 控制平面组件。
- 在 `host cluster` 上部署 Karmada 控制平面组件。
- 创建成员集群并加入 Karmada。

如果一切顺利，在脚本输出的末尾，您将看到类似的消息，如下所示：

```
Local Karmada is running.

To start using your Karmada environment, run:
  export KUBECONFIG="$HOME/.kube/karmada.config"
Please use 'kubectl config use-context karmada-host/karmada-apiserver' to switch the host and control plane cluster.

To manage your member clusters, run:
  export KUBECONFIG="$HOME/.kube/members.config"
Please use 'kubectl config use-context member1/member2/member3' to switch to the different member cluster.
```

Karmada 有两个上下文：

- **Karmada-apiserver** `kubectl config use-context karmada-apiserver`
- **karmada-host** `kubectl config use-context karmada-host`

`karmada-apiserver` 是与 Karmada 控制平面交互时使用的主要 kubeconfig，而 `karmada-host` 仅用于调试主机集群上的 Karmada 安装。您可以通过运行 `kubectl config view` 随时检查所有集群。要切换集群上下文，请运行 `kubectl config use-context [CONTEXT_NAME]`。

请注意，虽然我们有一个新的上下文 `karmada-apiserver`，但这不是实际的 Kubernetes 集群。相反，它是在 `karmada-host` Kubernetes 集群内运行的 Karmada 控制平面 API 服务器。

要列出在 `karmada-host` 集群上运行的 Karmada 控制平面组件，请运行以下命令。

```
$ kubectl config use-context karmada-host
$ kubectl get pod -n karmada-system
```

要列出添加到 `karmada-apiserver` 的目标集群，请执行以下命令：

```
$ kubectl config use-context karmada-apiserver
$ kubectl get clusters
```

`karmada-apiserver` 的目标中将会有三个集群，分别是 `member1`、`member2` 和 `member3`。

如果我们想要将资源联合到主机集群以及 `member1`、`member2` 和 `member3`（我们不会在演示中执行此操作），我们可以使用以下命令将此 `karmada-host` 添加为目标集群到 `karmada-apiserver`。

```
# Do not run this for the demo!
$ karmadactl join host \
  --karmada-context=karmada-apiserver \
  --cluster-context=karmada-host
```

使用 Karmada 部署多集群应用程序

现在我们将使用 Karmada 将示例 nginx 应用程序部署到多个集群中。

1. 将 kube 上下文更改为 `karmada-apiserver`

```
$ kubectl config use-context karmada-apiserver
```

2. 在 Karmada API Server 中创建 nginx 部署

现在，我们将在 Karmada API 服务中创建一个 Nginx 部署。它将创建资源，但不会将 nginx 部署传播到任何成员集群，直到我们应用 PropagationPolicy。

```
$ kubectl create -f nginx-deployment.yaml
```

Nginx 部署清单如下：

```
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
```

3. 创建将 nginx 传播到成员集群的 PropagationPolicy

Karmada 在 PropagationPolicy 中提供了大量配置以允许各种传播策略。我们现在将讨论它们。

3.1 复制/重复多集群Nginx部署

在此复制的多集群 PropagationPolicy 中，nginx 部署将复制到所有集群。因此，当我们应用下面的 PropagationPolicy 时，所有成员集群都将部署带有两个副本的 nginx。

请注意，在下面的 PropagationPolicy 中，我们将字段设置为 `replicaSchedulingType: Duplicated`


```
apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: nginx-propagation
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: nginx
  placement:
    replicaScheduling:
      replicaSchedulingType: Duplicated
```

```
$ kubectl create -f propagationpolicy.yaml
```

如果我们查看部署状态，我们将看到总共有六个副本正在运行（每个成员集群有两个副本）。

```
(* | karmada-apiserver:default) ~ k get deployments.apps nginx
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx	6/2	6	6	50m

 云原生拓展

3.2 分割多集群Nginx部署

在划分多集群传播策略中，nginx 部署副本将划分到所有成员集群上。我们还可以配置跨成员集群的副本分配权重。

请注意，在下面的 PropagationPolicy 中，我们将该字段设置为 `replicaSchedulingType: Divided` 并给出了副本分布权重首选项。

```

apiVersion: policy.karmada.io/v1alpha1
kind: PropagationPolicy
metadata:
  name: nginx-propagation
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: nginx
  placement:
    replicaScheduling:
      replicaDivisionPreference: Weighted
      replicaSchedulingType: Divided
      weightPreference:
        staticWeightList:
          - targetCluster:
              clusterNames:
                - member1
              weight: 1
          - targetCluster:
              clusterNames:
                - member2
              weight: 1

```


```
$ kubectl create -f propagationpolicy.yaml
```

如果我们查看部署状态，我们将看到总共有两个副本分布在 `member1` 和 `member2` 集群上，这意味着这两个集群都有一个单独的副本。

下图显示了 `karmada-apiserver` 上的部署状态。

```
(* | karmada-apiserver:default) → ~ k get deployments.apps nginx
```


NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx	2/2	2	2	75m

 云原生拓展

下图显示了 `member1` 集群上的部署状态。

```
(* | member1:default) → ~ k get deployments.apps nginx
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx	1/1	1	1	78m

 云原生拓展

3.3 仅将部署传播到选定的集群

在传播策略中，我们可以选择希望传播工作负载的成员集群。这可以通过 Karmada 在 `ClusterAffinity` 下提供的各种选项来实现。这些选项是：

- LabelSelector
- FieldSelector
- ClusterNames
- Exclude

在这里，我将给出一个基于 ClusterNames 的示例传播策略。

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

```
$ kubectl create -f propagationpolicy.yaml
```

如果我们检查部署的状态，我们将看到 Karmada 仅将部署传播到 `member1` 集群，并且两个副本都仅在那里运行。

下图显示了 karmada-apiserver 上的部署状态。

```
(* | karmada-apiserver:default) → ~ k get deployments.apps nginx
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx	2/2	2	2	82m

云原生拓展

下图显示了 `member1` 集群上的部署状态。

```
(* | member1:default) → ~ k get deployments.apps nginx
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx	2/2	2	2	82m

云原生拓展

3.4 覆盖策略以允许覆盖每个集群的配置

OverridePolicy 用于声明资源传播到不同集群时的覆盖规则。

现在，在覆盖策略的帮助下，我们将在 `nginx` 部署上添加一个新标签 `env: member1`，仅传播到集群 `member1`。如果我们查看 `nginx` 部署的当前状态，我们将看到 Karmada 默认添加的一些标签。

```
(* member1:default)* ~ k get deployments.apps nginx --show-labels
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
nginx   2/2     2            2           89m   app=nginx,propagationpolicy.karmada.io/name=nginx-propagation,propagationpolicy.karmada.io/namespace=default,resourcebinding.karmada.io/key=5bdc5c4dcf,work.karmada.io/name=nginx-687f7fb96f,work.karmada.io/namespace=karmada-es-member1
```

我们将应用以下覆盖政策。

```
apiVersion: policy.karmada.io/v1alpha1
kind: OverridePolicy
metadata:
  name: nginx-op
spec:
  resourceSelectors:
    - apiVersion: apps/v1
      kind: Deployment
      name: nginx
  overrideRules:
    - targetCluster:
        clusterNames:
          - member1
      overrides:
        labelsOverride:
          - operator: add
            value:
              env: member1
```

```
$ kubectl create -f overridepolicy.yaml
```

现在，如果我们在 `member1` 集群中看到 `nginx` 部署的状态，我们应该会看到那里添加了一个新标签 `env: member1`。

```
(* member1:default)* ~ k get deployments.apps nginx --show-labels
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   LABELS
nginx   2/2     2            2           94m   app=nginx,env=member1,propagationpolicy.karmada.io/name=nginx-propagation,propagationpolicy.karmada.io/namespace=default,resourcebinding.karmada.io/key=5bdc5c4dcf,work.karmada.io/name=nginx-687f7fb96f,work.karmada.io/namespace=karmada-es-member1
```

总结

在 Karmada 的帮助下，可以编排多集群、多云部署。Karmada 支持各种选项来传播目标集群中的资源，可以根据需要使用这些资源。覆盖策略非常方便地将特定于集群的配置应用于工作负载。

Karmada 提供了本文中未涵盖的许多其他有用功能。以下是 Karmada 官方文档（<https://karmada.io/docs/>）中进一步阅读的一些内容。

- Karmada 处理目标集群故障转移
- 全局资源搜索
- Descheduler 用于重新编排
- 用于重新调度的集群准确调度程序估计器
- 基于集群资源建模的调度
- 多集群服务发现
- 多集群 Ingress

