作者：李晓辉

联系方式：

1. 微信：Lxh_Chat

2. 邮箱：939958092@qq.com

管理资源清单

Kubernetes 集群里的应用一般都是好几个资源一起配合工作的，每个资源都有自己的定义和配置。搞定这些定义和配置，主要有两种方法：

1. **命令式**：这种方式就像直接给 Kubernetes 下命令一样，比如用 `kubectl create`、`kubectl delete` 这些命令来直接操作资源。这种方法比较直接，适合快速操作，但要是资源多了，管理起来就有点麻烦，容易乱。
2. **声明式**：这种方式就更高级一点，你先写一个 YAML 文件，把资源的最终状态描述清楚，然后用 `kubectl apply` 命令让 Kubernetes 去实现这个状态。不管资源现在是什么样子，Kubernetes 都会自动调整，让它变成你描述的那个样子。这种方法更适合复杂的系统，管理起来更方便，也更容易维护。

简单来说，命令式就是直接动手操作，声明式就是告诉 Kubernetes 你想要的结果，让 Kubernetes 自己去搞定。

使用命令式管理

Kubernetes CLI 使用命令式和声明式命令。命令式命令执行基于命令的操作，并且使用与操作密切相关的命令名称

例如：以下为命令式创建一个pod

```
[student@workstation ~]$ oc login -u admin -p redhatocp https://api.ocp4.example.com:6443
Login successful.

[student@workstation ~]$ oc run nginx --image=registry.ocp4.example.com:8443/ubi9/httpd-24:
```

命令式操作虽然有点小毛病，但也挺有用的！

首先，命令式操作确实有点小问题。比如，它不太好重复操作，每次手动敲命令，很容易出错。而且，它没有版本控制，一旦改错了，很难回溯。还有，它也不太支持 GitOps，现在大家都喜欢用 GitOps 来管理配置，命令式操作就有点跟不上节奏了。

但是，命令式操作在调试的时候可太有用了！比如在测试环境里，你可以快速给 Deployment 设置一个新的镜像，或者加一个新的环境变量，甚至直接跳进 Pod 里去操作。这些操作虽然简单，但有时候能快速解决问题，特别适合临时调试用。

使用声明式管理

命令式虽然有时候挺方便的，但真的有点小毛病。相比之下，声明式那才是真香！

声明式操作是通过资源清单来管理资源的，这才是正经路子。**资源清单就是那种 JSON 或 YAML 格式的文件，里面写满了资源的定义和配置信息**。这些清单文件把应用的所有属性都打包在一个文件或者一组文件里，这样一来，管理 Kubernetes 资源就简单多了。Kubernetes 只要读取这些资源清单，然后把清单里定义的状态应用到集群里就行。

而且，资源清单用的是 YAML 或 JSON 格式，这可太有用了！**它们能做版本控制**，你可以把配置文件放到 Git 仓库里，每次修改都能留下痕迹。要是哪次改坏了，直接回滚到之前的版本就行，这可比命令式方便多了。

最厉害的是，**资源清单能确保应用的精确再现**。你可以用一个命令就把一大堆资源都部署好，而且每次部署的结果都一样。这种可再现性特别适合持续集成和持续开发（CI/CD）的 GitOps 实践，自动化起来超方便。

举例来说，以下yaml创建了一个名为lixiaohui的pod，而yaml文件本身是源代码，可以放到git中，用于版本控制和再现

```
cat > pod.yml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: lixiaohuipod
spec:
  containers:
  - name: hello
    image: registry.ocp4.example.com:8443/ubi9/httpd-24:latest
    imagePullPolicy: IfNotPresent
  restartPolicy: OnFailure
EOF
```

```
oc apply create/apply -f pod.yml/URL/DIR
```

1. create并不会检查线上资源是否存在，创建资源时如果线上存在同名的资源，会报错
2. apply在创建资源时如果线上存在同名的资源，会用yaml中的参数对线上资源进行更改，如果线上不存在同名资源，则自动执行create操作

k8s yaml小技巧

你可能会觉得上面的yaml较为复杂，结构不清晰，这在初学的时候很正常，以下是了解结构的好方法：

dry-run

用 `--dry-run` 和 `-o yaml` 这两个选项，就能把命令式操作转换成对应的 YAML 清单。这样就能把临时的命令式操作变成可以保存和版本控制的声明式配置啦。

讲讲 `--dry-run=client` 和 `--dry-run=server` 的区别，这两个选项在 Kubernetes 命令中用来模拟操作，但它们的执行方式和作用范围有所不同。

`--dry-run=client`

- **执行位置**：在客户端（即你的本地机器上）模拟操作。
- **作用**：
  - **不与集群通信**：命令不会发送到 Kubernetes 集群，只是在本地生成资源的 YAML 或 JSON 表示。
  - **生成配置文件**：非常适合用来生成资源清单文件，比如 YAML 文件。你可以用它来查看命令的输出，然后将输出保存为文件，后续可以用 `kubectl apply` 来应用。
  - **快速验证**：可以快速验证命令的语法是否正确，而不会对集群产生任何实际影响。
- **使用场景**：
  - 生成资源清单文件。
  - 快速验证命令语法。
  - 在本地调试配置，而不影响集群。

`--dry-run=server`

- **执行位置**：在 Kubernetes 集群的服务器端模拟操作。
- **作用**：
  - **与集群通信**：命令会发送到 Kubernetes 集群，集群会处理这个命令，但不会真正应用更改。
  - **验证集群状态**：可以验证命令在当前集群状态下的行为，包括资源的创建、更新或删除。
  - **检查权限**：可以检查你是否有足够的权限执行该操作。
  - **检查冲突**：可以检查资源是否存在冲突，例如是否已经有相同名称的资源。
- **使用场景**：
  - 在集群环境中验证命令的效果。
  - 检查权限和资源冲突。
  - 在实际应用更改之前，确保命令在集群中可以正常执行。
- **`--dry-run=client`**：在本地模拟，不与集群通信，适合生成配置文件和快速验证语法。
- **`--dry-run=server`**：在集群服务器端模拟，与集群通信，适合验证命令在集群环境中的行为，检查权限和冲突。

```
[student@workstation ~]$ oc create deployment nginx --image=registry.ocp4.example.com:8443/
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"kind":"Deployment","apiVersion":"apps/v1","metadata":{"name":"nginx","creationTimes
  creationTimestamp: null
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
      - image: registry.ocp4.example.com:8443/ubi9/httpd-24:latest
        name: httpd-24
        resources: {}
status: {}
```

以下命令将dry-run设置为server，你就可以在不真的创建的前提下，去测试你的命令或yaml服务器是否可以接受并创建

```
[student@workstation ~]$ oc run nginx-2 --image=registry.ocp4.example.com:8443/ubi9/httpd-2
pod/nginx-2 created (server dry run)
```

还需要注意`--save-config`选项

当你在创建或更新 Kubernetes 资源时，使用`--save-config`选项，Kubernetes 会将你通过命令行指定的配置保存到资源的注解中。这些注解会存储在资源的`metadata.annotations`字段中，通常以`kubectl.kubernetes.io/last-applied-configuration`为键。

YAML的图形化视图

打开Web控制台--工作负载--Pod--创建pod

Explain

`explain` 命令在 Kubernetes 和 OpenShift 中超级有用！它能帮你深入了解资源清单中任何字段的详细信息，就像是一个在线的“字段说明书”。

你可以用 `kubectl explain` 或 `oc explain` （在 OpenShift 环境中）来查看资源清单中各个字段的详细描述。比如你想了解 Deployment 的某个字段，就可以用这个命令`oc explain deployment.spec.template.spec` 。

下面的例子查看了pod以及pod下一级的字段列表和解释

```
[student@workstation ~]$ oc explain pod
[student@workstation ~]$ oc explain pod.metadata
```

diff

`oc diff` 是一个超实用的命令，它可以帮助你快速对比线上资源和本地 YAML 文件之间的差异。这在开发和运维过程中特别有用，能让你清楚地看到哪些地方有变化，避免不小心改错了配置。

我们先来创建一个deployment

```
cat > deployment.yml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: lixiaohui-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: registry.ocp4.example.com:8443/ubi9/httpd-24:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
EOF
```

```
[student@workstation ~]$ oc apply -f deployment.yml
```

完成创建后，我们稍微修改以下deployment.yml文件内容，比如将镜像改为nginx，之后我们运行oc diff

```
[student@workstation ~]$ oc diff -f deployment.yml
Warning: would violate PodSecurity "restricted:latest": allowPrivilegeEscalation != false (
diff -u -N /tmp/LIVE-394511264/apps.v1.Deployment.default.lixiaohui-deployment /tmp/MERGED-
--- /tmp/LIVE-394511264/apps.v1.Deployment.default.lixiaohui-deployment 2025-05-08 12:33:26
+++ /tmp/MERGED-3914567848/apps.v1.Deployment.default.lixiaohui-deployment      2025-05-08
@@ -4,7 +4,7 @@
   annotations:
     deployment.kubernetes.io/revision: "1"
   creationTimestamp: "2025-05-08T04:32:50Z"
-  generation: 1
+  generation: 2
   labels:
     app: nginx
   name: lixiaohui-deployment
@@ -30,7 +30,7 @@
         app: nginx
     spec:
       containers:
-      - image: registry.ocp4.example.com:8443/ubi9/httpd-24:latest
+      - image: nginx
         imagePullPolicy: IfNotPresent
         name: nginx
         ports:
```

从输出看，很容易能看出我们如果apply的话，会发生哪些变化，非常实用吧~

本文档在线版本：https://www.linuxcenter.cn