

作者：李晓辉

联系方式：

1. 微信：Lxh_Chat

2. 邮箱：939958092@qq.com

你知道 Kustomize 吗？超好用的配置管理工具！咱平时创建 deployment、service 这些资源，不就是改镜像、名称、标签啥的嘛。以前每次都得写一长串 YAML，麻烦得很。现在有了 Kustomize，咱们先写好常用的 YAML 模板，以后有新需求，直接用它把新参数覆盖到模板里，就能搞定新部署啦，再也不用重复造轮子，省心多了！

Kustomize 的文件结构主要围绕 **base** 和 **overlay** 两个核心概念展开。每个 Kustomization 文件夹（无论是 base 还是 overlay）都必须包含一个 `kustomization.yaml` 文件，这个文件是 Kustomize 的核心配置文件。

- `base/` 文件夹存放通用的基础模板。
- `overlays/` 文件夹存放针对不同环境（如开发环境 `dev` 和生产环境 `prod`）的覆盖配置。

Base 目录

Base 是通用模板，包含了所有资源的通用部分。`base` 目录通常包含以下内容：

1. **资源文件**：这些是 Kubernetes 资源的 YAML 文件，比如 `deployment.yaml`、`service.yaml` 等。
2. **`kustomization.yaml` 文件**：这个文件定义了如何组合和管理这些资源文件。

`kustomization.yaml` 文件的作用

`kustomization.yaml` 文件在 `base` 目录中的作用主要包括以下几点：

1. **列出资源文件**：通过 `resources` 字段，指定 `base` 目录中包含的所有资源文件。
2. **定义通用配置**：可以定义一些通用的配置，比如通用的标签（`commonLabels`）、注解（`commonAnnotations`）等。
3. **定义生成的资源名称前缀或后缀**：通过 `namePrefix` 或 `nameSuffix` 字段，为所有资源名称添加前缀或后缀。

下面是base目录的基本结构：

```
base
├─ configmap.yaml
├─ deployment.yaml
├─ secret.yaml
├─ service.yaml
├─ route.yaml
└─ kustomization.yaml
```

base目录中的 `kustomization.yaml` 文件内容示例如下：

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- configmap.yaml
- deployment.yaml
- secret.yaml
- service.yaml
- route.yaml
```

base案例

创建出base目录，并在目录中，创建出通用的模板以及kustomization.yaml

```
mkdir base
cd base
```

```
cat > Secret.yml <<--EOF
apiVersion: v1
data:
  password: QUJDYWJjMTIz
kind: Secret
metadata:
  name: mysqlpass
EOF
```

```
cat > Deployment.yml <<-EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: mysqlname
          image: registry.ocp4.example.com:8443/rhel8/mysql-80:latest
          imagePullPolicy: IfNotPresent
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysqlpass
                  key: password
EOF
```

```
cat > Service.yml <<-EOF
apiVersion: v1
kind: Service
metadata:
  name: nodeservice
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 80
      nodePort: 31788
EOF
```

最后创建出kustomization.yaml，在这个文件中，我们包含上我们刚创建的3个资源

```
cat > kustomization.yaml <<-EOF
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
metadata:
  name: lxh-base-kustomization
resources:
- Secret.yml
- Deployment.yml
- Service.yml
EOF
```

查看一下目前base目录的结构

```
[student@workstation base]$ pwd
/home/student/base
[student@workstation base]$ tree .
.
├── Deployment.yml
├── kustomization.yaml
├── Secret.yml
└── Service.yml

0 directories, 4 files
```

以上在base目录中的文件将用于生成：

1. 一个名为mysqlpass的机密
2. 一个名为nginx-deployment的部署，此部署的pod将具有app: nginx标签，并引用mysqlpass机密作为密码，密码值为ABCabc123
3. 一个名为nodeservice的服务，监听在8000端口，收到请求后，转发给具有app: nginx标签的pod，并启用了31788的nodePort

至此，我们的三种类型的通用模板就完成了，可以开始overlay的参数覆盖部分了

overlay目录

Kustomize 可以通过覆盖（Overlay）来修改基础（Base）里的配置，但完全不会动到基础里的原始文件。这就像是给基础模板“穿了个外套”，只改变外表，内核还是那个内核。

每个覆盖目录里都有个 `kustomization.yaml` 文件，这个文件就像是指挥官，它能指定一个或多个基础目录。也就是说，你可以有一个通用的基础模板，然后针对不同的需求（比如开发环境、测试环境、生产环境）创建不同的覆盖目录。每个覆盖目录都可以用同一个基础目录，然后通过自己的 `kustomization.yaml` 文件来定义怎么修改基础里的东西，比如改改镜像版本、副本数量之类的。

举个例子，你有个基础模板，里面定义了一个应用的基本配置。然后你创建了一个开发环境的覆盖目录，在这个目录的 `kustomization.yaml` 文件里，你可以指定用那个基础模板，然后再加上一些开发环境特有的配置，比如把副本数量改成 2，或者给资源名称加上 `dev-` 前缀。这样一来，开发环境就有了自己的配置，但基础模板还是原封不动的。

这样做的好处是，不管你有多个环境，基础模板都保持一致，方便管理和更新。而且，每个环境的覆盖目录都是独立的，互不干扰，想改哪里就改哪里，超灵活！

具体来说，加入overlay之后的文件结构如下：

```
[student@workstation ~]$ tree .
.
├── base
│   ├── configmap.yaml
│   ├── deployment.yaml
│   ├── secret.yaml
│   ├── service.yaml
│   ├── route.yaml
│   └── kustomization.yaml
├── overlay
│   ├── development
│   │   └── kustomization.yaml
│   ├── testing
│   │   └── kustomization.yaml
│   └── production
│       ├── kustomization.yaml
│       └── patch.yaml
```

overlay的 `kustomization.yaml` 文件中必须写明，对哪类资源做哪些必要的更改

overlay案例

创建开发环境

kustomization

```
cd
mkdir -p overlays/development
cd overlays/development
```

创建开发环境的kustomization.yaml 文件：

Kustomize 功能特性列表参阅：

1. patchesStrategicMerge补丁将会更新nginx-deployment这个Deployment
2. patchesJson6902补丁也会更新nginx-deployment这个Deployment
3. overlay的对象是刚创建的base目录下的内容
4. 全体对象添加env: dev
5. 禁止添加hash后缀
6. 产生两个新的configmap和secret

```
cat > kustomization.yaml <<-EOF
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: lxh-dev
patches:
- path: patchesStrategicMerge-demo.yaml
  target:
    kind: Deployment
    name: nginx-deployment
  options:
    allowNameChange: true
- path: patchesJson6902-demo.yaml
  target:
    kind: Deployment
    name: nginx-deployment
  options:
    allowNameChange: true
resources:
- ../../base
commonLabels:
  env: dev
generatorOptions:
  disableNameSuffixHash: true
configMapGenerator:
- name: cmusername
  files:
    - configmap-1.yml
- name: cimage
  literals:
    - cimage=18
secretGenerator:
- name: username
  files:
    - secret-1.yml
  type: Opaque
- name: secrettest
  literals:
    - password=LiXiaoHui
  type: Opaque
EOF
```

生成器Generator

在kustomization.yaml，如果用文件来生成configmap和secret，会将文件名也作为数据的一部分，建议用 literals

生成configmap和secret的文件

```
cat > configmap-1.yml <<-EOF
username=lixiaohui
EOF
```

```
cat > secret-1.yml <<-EOF
username=admin
password=secret
EOF
```

策略性合并与JSON补丁

在 Kustomize 里，`patchesStrategicMerge` 和 `patchesJson6902` 都是用来改 Kubernetes 资源的，但它们的风格和用法有点不一样。

1. `patchesStrategicMerge`

这种方式就是用 YAML 文件来改东西，特别直白。你可以直接在 YAML 文件里找到要改的地方，然后像平时写 Kubernetes 配置一样改它。如果你对 Kubernetes 的配置挺熟悉，用这个就特别顺手。比如，你想把一个 Deployment 的副本数从 1 改成 3，或者给 Pod 加个环境变量，直接在 YAML 里改就行，一眼就能看明白。

2. `patchesJson6902`

这个就有点高级了，用的是 JSON 补丁。它遵循一个叫 JSON Patch（RFC 6902）的规范，能干的事更多。你可以用它来添加、删除、替换字段，甚至还能测试字段的值。不过，它用的是 JSON 格式，看起来可能会比 YAML 稍微复杂一点。不过，要是你遇到一些特别复杂的修改，比如要根据条件动态改字段，或者要改嵌套很深的字段，`patchesJson6902` 就能大显身手了。

简单来说，`patchesStrategicMerge` 就像用笔在纸上直接改东西，直观又方便；`patchesJson6902` 就像用一把瑞士军刀，功能强大，但可能需要多花点时间熟悉怎么用。

生成策略性合并补丁

这里的名字一定要和已有的资源的名称一致

更新deployment的replicas为4


```
cat > patchesStrategicMerge-demo.yaml <<-EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
EOF
```

生成JSON补丁

新增deployment下的pod标签为dev: release1

```
cat > patchesJson6902-demo.yaml <<-EOF
[
  {
    "op": "add",
    "path": "/spec/template/metadata/labels/dev",
    "value": "release1"
  }
]
EOF
```

目前的文件列表：

```
[student@workstation development]$ tree .
.
├── configmap-1.yml
├── kustomization.yaml
├── patchesJson6902-demo.yaml
├── patchesStrategicMerge-demo.yaml
└── secret-1.yml

0 directories, 5 files
```

验证overlay最终成果

```
[student@workstation development]$ kubectl kustomize ./
```

输出

```
apiVersion: v1
data:
  cimage: "18"
kind: ConfigMap
metadata:
  labels:
    env: dev
  name: cimage
  namespace: lxh-dev
```

```
apiVersion: v1
data:
  configmap-1.yml: |
    username=lixiaohui
kind: ConfigMap
metadata:
  labels:
    env: dev
  name: cmusername
  namespace: lxh-dev
```

```
apiVersion: v1
data:
  password: QUJDYWJjMTIz
kind: Secret
metadata:
  labels:
    env: dev
  name: mysqlpass
  namespace: lxh-dev
```

```
apiVersion: v1
data:
  password: TGLYaWFvSHVp
kind: Secret
metadata:
  labels:
    env: dev
  name: secrettest
  namespace: lxh-dev
type: Opaque
```

```
apiVersion: v1
data:
  secret-1.yml: dXNlcm5hbWU9YWRTaW4KcGFzc3dvcmQ9c2VjcmV0Cg==
kind: Secret
metadata:
```

```
  labels:
    env: dev
  name: username
  namespace: lxh-dev
type: Opaque
---
apiVersion: v1
kind: Service
metadata:
  labels:
    env: dev
  name: nodeservice
  namespace: lxh-dev
spec:
  ports:
  - nodePort: 31788
    port: 8000
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
    env: dev
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
    env: dev
  name: nginx-deployment
  namespace: lxh-dev
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      env: dev
  template:
    metadata:
      labels:
        app: new-label
        env: dev
    spec:
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
```

```
secretKeyRef:
  key: password
  name: mysqlpass
image: registry.ocp4.example.com:8443/rhel8/mysql-80:latest
imagePullPolicy: IfNotPresent
name: mysqlname
```

发布开发环境

```
kubectl create namespace lxh-dev
kubectl apply -k .
```

```
configmap/cmage created
configmap/cmusername created
secret/mysqlpass created
secret/secrettest created
secret/username created
service/nodeservice created
deployment.apps/nginx-deployment created
```

查询创建的内容

发现我们新configmap和secret已经生效，两个补丁也都生效了，一个补丁将deployment的pod数量该为4，一个补丁添加了dev=release1的标签

```
[student@workstation development]$ kubectl get configmaps -n lxh-dev
```

NAME	DATA	AGE
cmage	1	41s
cmusername	1	41s
kube-root-ca.crt	1	11m

```
[student@workstation development]$ kubectl get secrets -n lxh-dev
```

NAME	TYPE	DATA	AGE
mysqlpass	Opaque	1	47s
secrettest	Opaque	1	47s
username	Opaque	1	47s

```
[student@workstation development]$ kubectl get service -n lxh-dev
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nodeservice	NodePort	10.106.128.145	<none>	8000:31788/TCP	51s

```
[student@workstation development]$ kubectl get deployments.apps -n lxh-dev
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx-deployment	4/4	4	4	55s

```
[student@workstation development]$ kubectl get pod --show-labels -n lxh-dev
```

NAME	READY	STATUS	RESTARTS	AGE	LABELS
nginx-deployment-6f86fd678b-bv688	1/1	Running	0	64s	app=nginx,dev=releas
nginx-deployment-6f86fd678b-wpk49	1/1	Running	0	64s	app=nginx,dev=releas
nginx-deployment-6f86fd678b-wr94h	1/1	Running	0	64s	app=nginx,dev=releas
nginx-deployment-6f86fd678b-xxkbw	1/1	Running	0	64s	app=nginx,dev=releas

本文档在线版本: <https://www.linuxcenter.cn>